Cooperative Multi-Agent Path Finding

Nir Greshler

Cooperative Multi-Agent Path Finding

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering

Nir Greshler

Submitted to the Senate
of the Technion — Israel Institute of Technology
Heshvan 5782 Haifa October 2021

This research was carried out under the supervision of Prof. Nahum Shimkin, of the Faculty of Electrical & Computer Engineering, and Dr. Oren Salzman, of the faculty of Computer Science.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author's doctoral research period, the most up-to-date versions of which being:

Nir Greshler, Ofir Gordon, Oren Salzman, and Nahum Shimkin. Cooperative multi-agent path finding: Beyond path planning and collision avoidance. Accpeted to the 3rd IEEE International Symposium on Multi-Robot and Multi-Agent Systems (MRS), 2021.

The following paper has been published during my M.Sc. studies but is not a part of this work:

Guy Revach, Nir Greshler, Nahum Shimkin. Planning for Cooperative Multiple Agents with Sparse Interaction Constraints. In he online Proceedings of the 6th Workshop on Distributed and Multi-Agent Planning (DMAP) at ICAPS, pages 48-56, 2020.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

In this research, we introduce and study the *Cooperative Multi-Agent Path Finding (Co-MAPF)* problem, an extension to the classical MAPF problem, where cooperative behavior is explicitly incorporated. The classical MAPF problem deals with a group of agents that move in a shared environment. This problem is inherently cooperative, since each agent has to arrive at a goal location, without colliding with other agents in the group. However, in many real-world applications, agents that operate in a shared environment are often *heterogeneous* and may have a different set of abilities and restrictions. Therefore, in the Co-MAPF framework, achieving goals and completing tasks may not depend only on avoiding collisions between agents, but also on actively coordinating their actions. Simply put, we may want agents not just to "not interrupt" each other, but also help each other achieve their goals. We term this a *truly cooperative* setting. In this setting, a group of autonomous agents operate in a shared environment and have to complete *cooperative tasks* while avoiding collisions with the other agents in the group. To complete cooperative tasks, agents must collaborate and coordinate their high-level decisions. This introduces a significant computational challenge on top of path planning and collision avoidance. This extension naturally models many real-world applications, where groups of agents are required to collaborate in order to complete a given task. To this end, we formalize the Co-MAPF problem and introduce *Cooperative Conflict-Based Search (Co-CBS)*, a CBS-based algorithm for solving the problem optimally for a wide set of Co-MAPF problems. Co-CBS uses a cooperation-planning module integrated into CBS such that cooperation planning is decoupled from path planning. We suggest two improvements to Co-CBS that significantly improve its success rate. We also address the *Task Assignment (TA)* problem, which is NP-hard in this context. We propose to formulate the TA problem as a *Multi-Index Assignment Problem (MIAP)*, use an off-the-shelf algorithm to solve it, and show how to integrate in into Co-CBS. Finally, we present empirical results on several MAPF benchmarks demonstrating the properties of several variants of Co-CBS.

Chapter 1

Introduction

Automated planning and scheduling, also denoted as *AI planning*, is a branch of artificial intelligence and decision theory that concerns the realization of strategies or action sequences, for execution by intelligent agents such as autonomous robots and unmanned vehicles. A planning task is a search and optimization problem whose purpose is to synthesize an optimal plan. That is, a sequence of actions which optimizes some objective criteria and leads an agent from its initial state to a target state.

A *multi-agent system (MAS)* is a distributed system composed of multiple autonomous intelligent agents, that work in a shared environment and carry out tasks to achieve goals. In a fully-cooperative multi-agent system, all agents work towards achieving a common goal, and affect each other actions and decisions. Their achievements are measured as a team using some group objective.

Multi-agent planning (MAP) [Torreño et al., 2017] is the process of coordinating the decisions and actions in a MAS. The main aspect in a cooperative MAP problem is *coordination*, i.e., ensuring that the actions of all agents result in a a jointly optimal plan for the group. This problem is motivated by many real-world applications in a variety of domains, such as military [Cil and Mala, 2010], logistics [Gebser et al., 2018], and search-and-rescue [Kitano and Tadokoro, 2001]. In these problems, agents must coordinate their decisions to maximize their (joint) team value.

The *Multi-Agent Path-Finding (MAPF)* problem is a special and important type of multi-agent planning. In MAPF [Stern et al., 2019], we consider a group of autonomous mobile agents (or robots) that operate in a shared space. Each robot has a start location and a goal location it needs to arrive at. The task is to find paths for each agent in a group, from its start to its goal location, without colliding with each other. An example MAPF problem instance is presented in Figure 1.1. In this example we have two robots, in light and dark blue colors, moving on a 4-connected grid with some obstacles (depicted with black squares). Each robot needs to arrive at its corresponding flag. The task is to a plan a path for each robot to its goal, without colliding with the other robot. The solution for this problem (i.e., a set of two paths), is shown using blue dots.

While relevant to many real-world applications, such as warehouse automation [Wurman et al., 2008], autonomous vehicles [Dresner and Stone, 2008; Švancara et al., 2019] and robotics [Hönig et al., 2016], recent research in the field has focused on expanding the classical MAPF

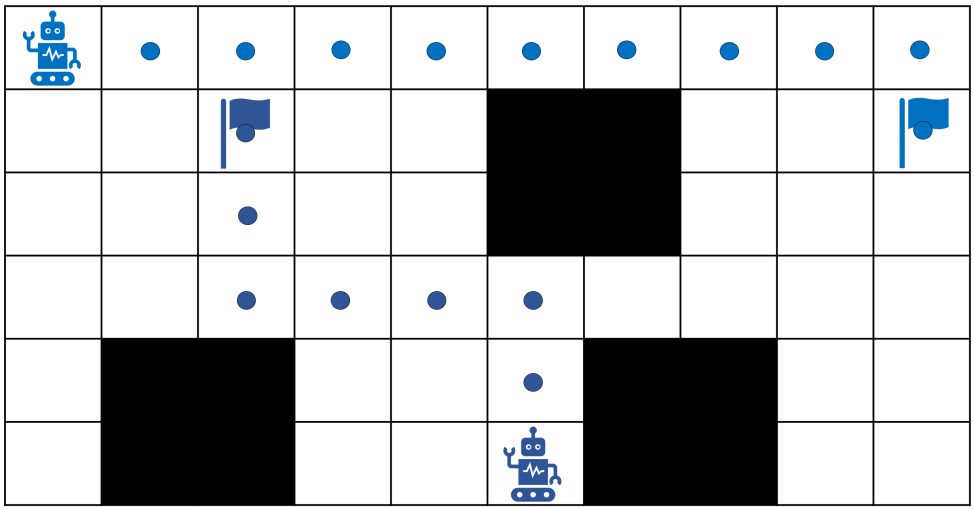


Figure 1.1: An example MAPF problem and solution.

framework to fit more real-world applications [Ma et al., 2017a; Felner et al., 2017; Salzman and Stern, 2020].

A main research direction towards the real-world applicability of MAPF is the problem of lifelong MAPF, also known as the Multi-Agent Pickup and Delivery (MAPD) problem. In this problem, a group of autonomous agents operate in a shared environment to complete a stream of incoming tasks, each with start and goal locations, while avoiding collisions with each others [Ma et al., 2017b; Liu et al., 2019]. A similar problem, studied by Ma et al. [Ma et al., 2016] is the package-exchange robot-routing problem (PERR) where payload exchanges and transfers are allowed, thus enabling the modelling of more general transportation problems.

The classical MAPF problem is inherently cooperative, since each agent has to arrive at its goal without colliding with other agents. However, in many real-world applications, agents that operate in a shared environment are often *heterogeneous* [Atzmon et al., 2020b] and may have a different set of abilities and restrictions. Furthermore, some tasks may be too complex for a single agent to complete on its own, and several agents of different types may have to work together. The main goal of this research is to investigate problems involving such tasks. More specifically, we are interested in a setting where agents have to coordinate their high-level decisions and plans, while not colliding with each other. We term this a *truly cooperative* setting. We wish to incorporate ideas from high-level multi-agent planning (for coordinating their high-level plans) into low-level path-finding (for avoiding collisions with each other). We focus on cases where high-level interactions between agents are *sparse*, and therefore can be represented in a compact way. More specifically, interactions among agents can be represented using constraints, such that collaborating agents are constrained to reach a specific interaction state at a specific time. The agents then must find an optimal plan that meets the constraints. This problem is investigated in a work [Revach et al., 2020] completed and published in the early stages of my M.Sc. studies. The full paper describing this work is presented in Appendix A for completeness.

To this end, we introduce the *Cooperative Multi-Agent Path Finding (Co-MAPF)* framework, a MAPF extension, in which a group of agents collaborate towards completing a joint task. In the

Figure 1.2: Autonomous robots transport inventory pods in an automated warehouse. *(Credit: https://www.greyorange.com/)*

Co-MAPF framework, achieving goals and completing tasks may not depend only on avoiding collisions between agents, but also on actively coordinating their actions. Simply put, we may want agents not just to "not interrupt" each other, but also help each other achieve their goals.

Our motivating problem is taken from the warehouse-automation domain [Wurman et al., 2008]. In this problem, storage locations host inventory pods that hold goods of different kinds. A large number of robots operate autonomously in the warehouse, picking up and carrying inventory pods (see Figure 1.2 for illustration) to designated drop-off locations, where goods are manually taken off the pods for packaging. The robots then carry the pods back to their location in the warehouse. In this scenario, the robots' main task is to transport the pods around the warehouse, and we refer to robots executing such tasks as *transfer units*. Incorporating robots to automate warehouses (such as those maintained by Amazon, Alibaba and more) has launched significant research on MAPF problems during the last decade, both in industry and academia. However, the problem is still far from being solved, as current solutions still assume a person picking up objects from shelves in a warehouse. Research in a different, yet closely-related area, has studied this exact problem—autonomous robotic arms capable of picking-up a specific item from an inventory pod [Correll et al., 2016]. We refer to a moving robot with such arm as a *grasp unit*. Our motivation in this research is to decouple the task of grasping an object and transporting it in the warehouse. We thus present an improved warehouse scenario, where robots of two types, grasp and transfer units, work together in coordination (for example, by scheduling a meeting between them) to improve some optimization objective.

This motivating example is depicted in Figure 1.3. Two pairs of robots operate in a warehouse – two grasp units and two transfer units. A grasp unit, depicted in red, is a mobile robot with a robotic arm, capable of picking a specific item or box from a shelf in the warehouse. A transfer unit, depicted in blue, is a light-weight, possibly cheaper mobile robot, that can carry items and boxes and transport them around the warehouse. Grasp unit #1 arrived at the task start location, i.e., next to the shelf. It will pick up the box and then drive to a meeting location

(marked with a yellow square) to transfer the box to transfer unit #1. The transfer unit has a path (marked with blue arrows) to the meeting point, and from there to the task goal (the P square), where the box will be picked by a human employee. The second pair of robots (#2 in Figure 1.3) are at their meeting location. They occupy the same location by following a special *rendezvous* protocol which allows them to safely transfer the box without colliding.

This improved scenario offers better flexibility and may significantly improve the warehouse performance. Instead of moving around the whole inventory pod back and forth in the warehouse, we can transport only the wanted item, by picking it up from the shelf using a robotic arm. By doing so, we can significantly reduce the delivery time of a single package, and reduce the number of actions needed by the robots, possibly achieving improved warehouse throughput and latency. We still need of course to make sure that robots do not collide with each other.

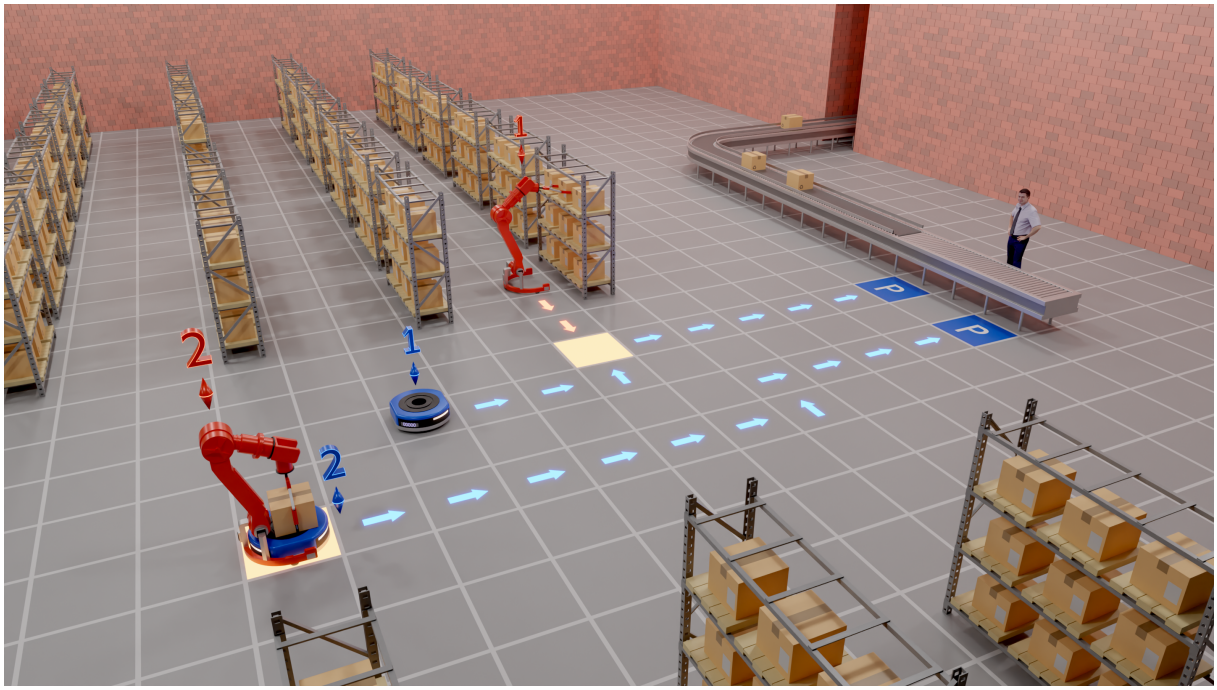


Figure 1.3: An improved warehouse scenario, where robots of two types collaborate to complete delivery tasks—the red robot, with a robotic arm, picks up a box from the shelf. Then the two robots meet and the box is transferred to the blue robot, that transports the box to its picking location. *(Credit: https://www.elitzurbaryehuda.com/)*

In this research, we suggest to incorporate a truly cooperative behavior to classical MAPF using the notion of *cooperative tasks*, such that agents coordinate their high-level decisions in the context of cooperative tasks, and avoid collisions in the low-level plans. Similar to (non-cooperative) tasks defined in the MAPD literature [Ma et al., 2017b; Liu et al., 2019], cooperative tasks are assigned to agents, rather than explicit goals. Agents are able to complete cooperative tasks (formally defined in Chapter 3) only by coordinating their actions and goals with each other.

In Chapter 3, we present the formulation of the Co-MAPF problem, define its input and solution. The formulation is derived from the classical MAPF formulation [Stern et al., 2019], which is also presented as an algorithmic background. In addition, we discuss differences and further extensions to the Co-MAPF framework which can be used towards achieving more coop-

erative capabilities in a MAPF problem. In the suggested formulation, presented in Chapter 3, there is more than one set of agents, possibly representing heterogeneous real-world agents, and we specifically focus on the case of two sets of agents, of two different types. Agents' high-level interaction is restricted to the form of *meetings*. More specifically, agents have to schedule a meeting location and time to complete a task. We also discuss other forms of agent interactions, and generalizations to the suggested formulation. We state that besides the aforementioned warehouse problem, more real-world problems can be modeled using the Co-MAPF framework, such as the involvement of aerial robots in fulfilment centers [Shome, 2021], the truck-and-drone "last-mile" delivery problem [Murray and Raj, 2020] and multi-drone delivery using transit networks [Choudhury et al., 2020].

Based on the suggested formulation, we introduce (in Chapter 4) *Cooperative Conflict-Based Search (Co-CBS)*, an optimal three-level algorithm that is heavily based on two previously-suggested optimal algorithms: the well-known Conflict-Based Search (CBS) [Sharon et al., 2015] for solving a classical MAPF problem, and the Conflict-Based Search with Optimal Task Assignment (CBS-TA) [Hönig et al., 2018] for solving the anonymous MAPF problem, where we also need to assign goals (or tasks) to each agent. We also introduce two improvements to the basic version of Co-CBS, one previously-suggested CBS improvement, and another which is unique to Co-CBS and exploits special characteristics in the problem. Many more CBS extensions and improvements exist, some of which can be immediately applied to Co-CBS, as we discuss in Chapter 7. A theoretical analysis of Co-CBS is presented, where we define the notion of *source-connected* Co-MAPF instances, and prove that Co-CBS is *complete* on these instances. We also prove that Co-CBS is *optimal* on every solvable Co-MAPF instance.

In Chapter 5, we present an important extension to our suggested problem. More specifically, we address the *task assignment (TA)* problem, in which tasks are not pre-assigned to agents, but we also wish to determine which agents are paired together and which task is assigned to them, such that the objective function is optimized. We show that the task assignment problem for the Co-MAPF is equivalent to a well-known combinatorial optimization problem called the *multi-index assignment problem (MIAP)*, which is NP-hard. We suggest to use an off-the-shelf solver and describe how to incorporate it into Co-CBS.

We present empirical results of running Co-CBS on several MAPF benchmarks. We show that it solves nontrivial problem instances (detailed in Chapter 6), and that our two suggested Co-CBS improvements significantly improve the algorithm's performance. We also present empirical results when also solving the task assignment problem. These results show that solving the task assignment problem may significantly reduce the cost of the obtained solutions, and moreover, it also improves the algorithm performance (in terms of run time).

Finally, in Chapter 7 we conclude the work and discuss some extensions and research directions. Specifically, we discuss possible improvements and suggestions for Co-CBS, some of which include applying previously-suggested CBS improvements into Co-CBS. In addition, we suggest several extensions for the Co-MAPF framework, towards making it more real-life applicable.

Chapter 2

Background and Related Work

The Co-MAPF framework introduced in this work is an extension of the classical multi-agent path finding problem, which is a branch of the more general field of multi-agent planning. In this research, we draw inspiration and build upon several fields of multi-agent planning and multi-agent path finding. We also address the task-assignment problem, a well-known problem in combinatorial optimization, that has also been studied in the context of MAPF. In this chapter we provide a literature survey on most of the fields Co-MAPF relates to. We review several papers and ideas this research is based upon. As the goal of this research is to extend the MAPF framework, we also present recent work that also extend and generalize the MAPF problem.

2.1 Multi-Agent Path Finding (MAPF)

Multi-Agent Path Finding (MAPF) is a type of cooperative multi-agent planning, where the task is to plan paths for a group of agents, where interactions between agents are restricted to *collision avoidance*. Namely, each agent has to travel from a start location to a goal location without colliding with other agents, and while optimizing some team objective function. MAPF has many real-life applications, and has been researched extensively both in academia and industry. MAPF has several definitions and assumptions, which are summarized in [Stern et al., 2019]. Many algorithms have been proposed to solve the MAPF problem [Felner et al., 2017]. In this section, we present a survey of the Conflict-Based Search (CBS, [Sharon et al., 2015]) algorithm, a state-of-the-art optimal MAPF solver, as our research relies on it. CBS also has many extensions and improvements, and we present some of the most important ones.

2.1.1 Conflict-Based Search (CBS)

CBS [Sharon et al., 2012a; 2015] is a **complete** and **optimal** algorithm for solving the multi-agent path finding problem. It is based on the idea that agents' interactions are restricted to collision avoidance, and that collisions between agents are resolved by imposing constraints. CBS is a two-level algorithm. At the high level, a search is performed on a tree based on conflicts between agents, called the *constraint tree (CT)*. Whenever a conflict (or collision) is found, corresponding to two agents a_1, a_2 being at the same location v at the same time t, the CT node

is split into two new nodes, and a different constraint is added to each node, for each conflicting agent. The first constraint forbidding a_1 to be at location v at time t while the second forbidding a_2 to be at location v at time t. At the low level, a search is performed only for a single agent at a time, while satisfying all the imposed constraints. In many cases this reformulation enables CBS to examine fewer states than A* while still maintaining optimality. The performance of CBS depends on the structure of the problem. In cases with bottlenecks CBS performs well, and in open spaces CBS performs poorly. CBS outperforms other algorithms in cases where corridors and bottlenecks are more dominant.

2.1.2 Optimal CBS Variants

Many variants to CBS exist, that enhance its search technique, while keeping the algorithm optimal. We present some of the important optimal CBS variants.

Meta-Agent CBS (MA-CBS) [Sharon et al., 2012b]. MA-CBS is an optimal CBS variant that focuses on cases where CBS's run time is much slower than that of A*'s. The main idea is to couple groups of agents into meta-agents if the number of internal conflicts between them exceeds a given bound. MA-CBS acts as a framework that can run on top of any complete MAPF solver. In MA-CBS the number of conflicts allowed at the high-level phase between any pair of agents is bounded by a predefined parameter B. When the number of conflicts exceeds B, the conflicting agents are merged into a meta-agent and then treated as a joint composite agent by the low-level solver. By bounding the number of conflicts between any pair of agents, the exponential worst-case of basic CBS is prevented.

Bypassing a conflict. Boyarski et al. [2015a] propose a *bypassing (BP)* mechanism to improve the run time of CBS. When a conflict is found, the algorithm first attempts to bypass the conflict by finding a path of the same cost that does not pass the conflict location at the conflict time. This avoids the need to perform a split in the search tree and add new constraints. If no bypass is found, the algorithm performs the split action and adding explicit constraints to avoid the conflict. The authors propose two variants that search for a bypass. In the first one, the algorithm peeks at either of the immediate children of the CT node, trying to adopt their paths. The second bypass method generalizes the first, by searching the tree below the CT node, considering only nodes with same cost, and trying to use their paths in the solution of the current CT node.

The Improved CBS (ICBS) [Boyarski et al., 2015b]. ICBS builds upon the meta-agent and bypass improvements and adds two new improvements. Each of these improvements is strongly tied to one of the former improvements (MA and BP) as follows:

1. *Merge and restart (MR).* In MA-CBS, agents are merged locally at each node of the search tree. Instead, when a decision to merge is made, the authors suggest to restart the search from scratch, with the new merged meta-agent treated as a single agent for the entire search tree.

2. *Prioritizing conflicts (PC).* MA-CBS (even with BP added) arbitrarily chooses on which conflict to split the high-level constraint tree. Poor choices may increase the size of the tree. To remedy this, ICBS prioritizes the conflicts according to three types: *cardinal*, *semi-cardinal* and *non-cardinal*. Cardinal conflicts always cause an increase in the solution cost, so ICBS chooses to split cardinal conflicts first. Additionally, bypasses to cardinal conflicts cannot exist. Therefore, the optional BP improvement should only be applied for semi-cardinal or non-cardinal conflicts.

All four enhancements to CBS (i.e., MA, BP, PC and MR) are optional and can be added separately or in conjunction with the others, except for MR which is only relevant to MA-CBS. ICBS combines them all into a coherent improved version of CBS.

Iterative-Deepening Conflict-Based Search (IDCBS) [Boyarski et al., 2020]. IDCBS is a memory-bounded optimal variant of CBS that can be substantially faster than CBS due to incremental methods that it uses when processing CBS nodes. IDCBS replaces the high-level A*-like search of CBS with a search approach similar to ID A* [Korf, 1985], which is a search algorithm for exponential domains that uses memory conservatively. First, IDCBS explores the Conflict Tree (CT) using repeated depth-first iterations. Unlike A*, Depth-First Search (DFS) only moves from a parent node to its child and back and such nodes have many similarities in their content. Second, the authors identify six main components required to process a high-level CBS node and show how each one can be improved using incremental data structures that exploit similarities between parent nodes and their children. IDCBS is able to optimally solve many more problem instances than CBS and the authors report substantial improvements in search times even for problem instances which can be solved by a currently leading CBS variant.

2.1.3 Sub-optimal CBS Variants

CBS also have several sub-optimal variants. Barer et al. [2014] propose several CBS-based unbounded and bounded sub-optimal MAPF solvers. The proposed solvers relax the high and/or the low-level searches, allowing them to return a sub-optimal solution. *Greedy-CBS (GCBS)*, is a CBS-based MAPF solver designed for finding a (possibly sub optimal) solution as fast as possible, by preferring to expand search nodes that are more likely to produce a valid solution fast. *Bounded CBS (BCBS)* performs a *focal search* in both the low and high-level searches of CBS. Focal search maintains two lists of nodes: OPEN and FOCAL. The FOCAL list contains nodes with cost higher from the lowest-cost nodes in the OPEN list by some sub-optimality factor, which can also be selected for expansion. This ensures that the returned solution is within a given sub-optimality bound (which is the product of the bounds of the two search levels). *Enhanced CBS (ECBS)* is a bounded sub-optimal MAPF solver that also uses a focal search, but the high and low levels share a joint sub-optimality bound.

2.2 MAPF with Task Assignment

In MAPF with *Task Assignment* problem, the task is to first assign targets (or goals) to agents and then plan collision-free paths for the agents to their targets in a way such that the objective

function is optimized. Ma and Koenig [2016] study the *Task Assignment and Path Finding (TAPF)* problem for teams of agents. The agents are partitioned into teams, and each team has the same number of unique targets (goal locations). Any agent in a team can be assigned to a target of the team, and the agents in the same team are thus exchangeable. However, agents in different teams are not exchangeable. The task is to assign all goals to agents (within their teams) and plan collision-free paths, such that the *makespan* is minimized (i.e., the earliest time all agents arrive at their targets).

The TAPF problem is a generalization of the classical MAPF problem where tasks (or goals) are pre-assigned to agents, and the *anonymous* MAPF problem where we need to assign goals to agents. When only one team exists in TAPF, we get the anonymous MAPF problem, and when each team consists of exactly one agent, we get the classical (non-anonymous) MAPF problem. The anonymous MAPF problem can be solved optimally in polynomial time using *max-flow algorithm* on a time-expanded network [Yu and LaValle, 2012]. The non-anonymous MAPF problem (where the assignments of agents to target is pre-determined), is NP-hard and can be solved optimally using various algorithms, and CBS in particular.

Ma and Koenig [2016] propose the *Conflict-Based Min-Cost-Flow (CBM)*, a complete and optimal algorithm for solving the TAPF problem. CBM is a hierarchical algorithm that combines ideas from anonymous and non-anonymous MAPF algorithms. It uses CBS on the high level and a min-cost max-flow algorithm on a time-expanded network on the low level. In CBM, vertex and edge constraints are defined on a specific team (rather than agent). On the high level, CBM considers each team to be a meta-agent. It uses CBS to resolve collisions between meta-agents, that is, agents in different teams. On the low level, CBM uses a polynomial-time min-cost max-flow algorithm on a time-expanded network to assign all agents in a single team to unique targets of the same team and plan paths for them that obey the constraints imposed by the currently considered high-level node, and result in no collisions among the agents in the team.

The *Conflict-Based Search with Task Assignment (CBS-TA)* [Hönig et al., 2018] is proposed to solve the anonymous MAPF problem optimally for the *sum-of-costs* objective (i.e., the total time steps it takes all agents to arrive at their targets). CBS-TA extends CBS to jointly optimize task assignment and path planning. It operates on a search forest rather than a search tree, and creates the new search trees on demand. The algorithm is complete and optimal and the authors show that it outperforms methods that perform task assignment optimization and path planning independently. In CBS-TA, each node in the constraint tree contains a specific assignment of agents to goals and whether it is a root node (i.e., all nodes below the root node has the same assignment). Root nodes are expanded with the next-best assignment and added to the open list. The *Hungarian Algorithm* [Kuhn, 1955] is used to calculate the optimal assignment given a two-dimensional cost matrix and *Murty's Algorithm* [Murty, 1968] is used to generate the next best assignments. In the formulation presented in this paper, there are N agents at different start locations, and M potential goal locations, such that agents are permitted to have a set of possible goals. The binary $N \times M$ matrix A indicates whether an agent can be assigned to a specified goal. It is possible to model the aforementioned TAPF problem by setting $N = M$ and matrix A according to the group assignment. CBS-TA can compute optimal solutions with

respect to the sum of costs, which can be more relevant in some scenarios (e.g. minimizing the total energy usage of the team). It has also been shown that the makespan and sum-of-costs objectives cannot be simultaneously optimized. The authors also propose ECBS-TA, a bounded sub-optimal variant of CBS-TA, based on ECBS.

2.3 Multi-Agent Pickup and Delivery (MAPD)

The lifelong version of MAPF is called the *Multi-Agent Pickup and Delivery (MAPD)* problem. In the MAPD problem, agents have to attend to a stream of delivery tasks in an online setting. One agent has to be assigned to each delivery task. This agent has to first move to a given pickup location and then to a given delivery location while avoiding collisions with other agents.

The MAPD problem requires both the assignment of agents to tasks in an online and lifelong setting and the planning of collision-free paths. Since agents have to attend to a stream of tasks, they cannot rest in their destination (delivery) location after they finish executing tasks. Furthermore, in the online setting, tasks can enter the system at any time, therefore assigning agents to tasks and path planning cannot be done in advance but rather needs to be done during execution in real-time.

Ma et al. [2017b] formulate the MAPD problem and propose two algorithms to solve the problem. More specifically, we are given a set of unexecuted tasks $\mathcal{T}$, and in each time step, all new tasks are added to $\mathcal{T}$. Each task $\tau_i \in \mathcal{T}$ has a pickup location s_i and a delivery location g_i. An agent is called *free* iff it is not currently executing any task, and *occupied* otherwise. A free agent can be assigned to any task. When an agent arrives at a task pickup location, it starts to execute the task, and the task is removed from the task set. An agent can be assigned to a different task in the task set while it is still moving to the pickup location of the task it is currently assigned to but it has first to finish executing the task after it has reached its pickup location. When it reaches the delivery location, it finishes executing the task, which implies that it becomes a free agent and is no longer assigned to this task. A free agent can be assigned to any task in the task set. The objective is to finish executing each task as quickly as possible, that is to minimize the average number of time steps, called service time, needed to finish executing each task after it was added to the task set.

Two decoupled algorithms are proposed to solve this problem. The first is *Token Passing (TP)*. TP is based on an idea similar to Cooperative-A* [Silver, 2005]. It defines a *token*, which is a synchronized shared block of memory that contains the current paths of all agents, task set, and agent assignments. Any agent that has reached the end of its path in the token, requests the token once per timestep. The system then sends the token to each agent that requests it, one after the other. The agent with the token chooses a task from the task set such that no path of other agents in the token ends in the pickup or delivery location of the task:

- If there is at least one such task, the agent selects the one with a lowest heuristic value and plans collision-free path to pickup and then to delivery, and update the path in its token.
- If there is no such task, then the agent does not assign itself to a task in the current timestep. If the agent is not in the delivery location of a task in the task set, then it

updates its path in the token with the trivial path where it rests in its current location. Otherwise, to avoid deadlocks, it plans a collision-free path to an endpoint such that the delivery locations of all tasks in the task set are different from the chosen endpoint, and no path of other agents in the token ends in the chosen endpoint.

Finally, the agent returns the token to the system and moves along its path in the token.

The second algorithm, *Token Passing with Task Swaps (TPTS)*, is more effective than TP, by allowing agents to swap tasks. The set of tasks now contains all unexecuted tasks, rather than only all tasks that have no agents assigned. This means that an agent with the token can assign itself not only to a task that has no agent assigned but also to a task that is already assigned another agent as long as that agent is still moving to the pickup location of the task.

Liu et al. [2019] study an offline version of the MAPD version. In this setting, all tasks are known a-priori but have a certain release time, i.e., the time the task becomes available and can be assigned to an agent. The task assignment is determined by solving a *Traveling Salesman Problem (TSP)* where each node represents either an agent or a task release time. The output of solving the task assignment problem is a sequence of tasks for each agent, namely, the tasks each agent is assigned to, and the order of execution. Two algorithms are proposed to solve the offline MAPD problem. The first, *TA-Prioritized*, performs a prioritized planning. More specifically, after solving the task assignment, it plans for all agents one after the other, choosing the next agent whose path has the largest execution time. Once the chosen agent plan its path, all agents after it plan collision-free paths. To avoid deadlocks, TA-Prioritized uses a "reserving dummy path" mechanism. A dummy path of an agent is a path with minimal travel time to the parking location of the agent. The second algorithm is called *TA-Hybrid*. It considers two groups of agents for path planning and uses a different path-planning method for each group:

- *New-task* agents have to go from their current location to their delivery location, and cannot swap goals. Planning is done with ICBS.
- *Free* agents have to go from their current location to the pickup locations of the next task in their task sequence. Agents can swap pickup locations while moving. Planning is done with a min-cost max-flow algorithm to perform anonymous MAPF-based path planning.

After the task assignments, TA-Hybrid iterates over time steps, and checks if agents arrived at the delivery locations, and removes their task from the task sequence. Then it plans a path for all new-task agents to their delivery locations. Then it plans for all free agents (in $t = 0$ all agents are free) to their pickup locations. All agents then move one time step.

2.4 Extensions to Classical MAPF

Most MAPF-related work in recent years has focused on research directions that extended the classical MAPF problem to fit more real-world applications. In this section we discuss some of this work, as the main goal of this research is also to suggest an extensions to the classical MAPF problem.

Recall that our motivation problem, presented in Chapter 1, includes delivering a package by transferring it between two different robots. The problem of pickup and delivery with transfers

has already been studied in an online manner [Coltin and Veloso, 2014]. In this work, items need to be delivered from a pickup location to a delivery location within a time window, and robots are allowed to transfer items at arbitrary locations. Note that transfers are optional and cannot be enforced (as opposed to our motivation problem). Each time new requests arrive, or robots are stuck or delayed, the algorithm re-plans using an *auction* procedure, where it assigns new items to robots based on their bid, and adds transfers by using special actions *TransferSend* and *TransferReceive*. Several heuristics are used for the auction bids and transfers. Forcing the timing constraints (of item windows and transfers) is done using a *Simple Temporal Network (STN)*. Collisions between robots are not addressed directly in the planning phase. Instead, there may be delays in the execution of plans, which cause the algorithm to re-plan.

Ma et al. [2016] study the *package-exchange robot-routing problem (PERR)*, where robots are allowed to exchange payloads. PERR can be reduced to the integer multi-commodity network-flow problem and solved optimally using integer linear programming. It can also be converted to an adapted CBS where exchange operations (i.e., swapping conflicts) are allowed. The experiments show that flow-based solvers scale better on instances with many robots, however, the adapted CBS solver scale better on sparse grids with many bottlenecks. Note that in PERR, payloads exchange and transfers can help avoid bottlenecks and reduce the makespan. However, they cannot be enforced.

In the *Multi-Agent Meeting (MAM)* problem, the task is to find a meeting location for multiple agents, as well as a path for each agent to that location. It may be solved while considering conflicts between agents [Atzmon et al., 2021] or without [Atzmon et al., 2020a]. The goal is to minimize the makespan or sum of costs of all agents to the meeting location. Atzmon et al. [2020a] introduce a complete and optimal algorithm, *Multi-Directional Heuristic Search (MM*)* that finds the optimal meeting location under different cost functions and using several heuristic functions. Atzmon et al. [2021] address the *Conflict-Free Multi-Agent Meeting (CF-MAM)* problem, where the task is to find collision-free paths to the meeting location. Two optimal algorithms are presented to solve this problem.

Finally, many more challenged and generalization to the MAPF problem are discussed in several survey papers [Ma et al., 2017a; Felner et al., 2017; Salzman and Stern, 2020].

Chapter 3

Cooperative Multi-Agent Path Finding (Co-MAPF)

3.1 Introduction

In this chapter we describe and formally define the Cooperative Multi-Agent Path Finding (Co-MAPF) problem. The suggested formulation extends the classical MAPF formulation and adds high-level interactions between agents via the notion of *heterogeneous* agents that work together in collaboration, to complete *cooperative tasks*. More specifically, collaborating in the context of cooperative tasks is restricted to the form of *meetings*.

We first describe and formulate the classical MAPF problem followed by a formulation of our proposed Cooperative-MAPF (Co-MAPF) framework. We define the input for the problem, valid solutions, and define the objective function used to measure valid solutions.

Finally, we define the notion of *source-connected* Co-MAPF instances, which will allow us to check whether an instance is solvable in an efficient manner.

3.2 Classical Multi-Agent Path Finding (MAPF)

We first describe the classical MAPF problem in detail. Most of the material in this section is based on [Sharon et al., 2015] and [Stern et al., 2019].

3.2.1 Problem input

The input to the classical MAPF problem is:

1. A graph $G = (V, E)$ whose vertices V correspond to locations and whose edges E correspond to connections between the locations that the agents can move along.

2. A set of k *homogeneous* agents $A = \{a_1, \ldots, a_k\}$. Every agent a_i has a unique start vertex $s_i \in V$ and a goal vertex $g_i \in V$.

Time is discretized into time points $\{t_0, t_1, \ldots\}$, such that at time point t_0 agent a_i is located in location s_i.

3.2.2 Actions

In every time step, each agent is situated in one of the graph vertices and can perform a single action. Each agent has two types of actions: *move* and *wait*.

i) A move action means that the agent moves from its current vertex v to an adjacent vertex v' such that $(v, v') \in E$.

ii) A wait action means that the agent stays in its current vertex another time step.

Given a sequence of single-agent actions, p_i is called the *plan* or *path* of agent a_i, such that $p_i[t]$ is the location of agent a_i at time step t and $|p_i|$ is the length of the path. We denote $\mathcal{P} = \{p_1, \ldots, p_k\}$ the set of paths, one for each agent.

3.2.3 MAPF conflicts

A *conflict* in a MAPF problem represents a collision between two single-agent plans. The literature on classical MAPF includes several conflict definitions [Stern et al., 2019]. In this work we focus on two types of conflicts:

i) A *vertex conflict* between agents occurs when two agents occupy the same vertex at the same time. Formally, a vertex conflict between agents a_i and a_j exists iff there exists a time step t such that $p_i[t] = p_j[t]$.

ii) An *edge conflict* (sometimes called *swapping conflict*) between agents occurs when two agents traverse the same edge from opposite sides ("swap positions") at the same time step. Formally, an edge conflict between agents a_i and a_j exists iff there exists a time step t such that $p_i[t] = p_j[t+1]$ and $p_i[t+1] = p_j[t]$.

3.2.4 MAPF solution

A feasible (or valid) MAPF solution is a set of paths $\mathcal{P} = \{p_1, \ldots, p_k\}$ such that p_i is a path for agent a_i from vertex s_i to vertex g_i and there are no conflicts between any two paths in $\mathcal{P}$.

An optimal MAPF solution is a feasible set of paths $\mathcal{P}$ which optimizes some objective function (specifically defined in the next paragraph).

3.2.5 Objective functions

Arguably, the most common objective functions used in classical MAPF to evaluate solutions are *makespan (MKSP)* and *sum-of-costs (SOC)*, both to be minimized.

- **Makespan** is defined as the number of time steps required for all agents to reach their target. The makespan of a MAPF solution $\mathcal{P}$ is defined as: $\max_{1 \leq i \leq k} |p_i|$.

- **Sum-of-costs** is defined as the sum of time steps required by each agent to reach its goal. The sum of costs of a MAPF solution $\mathcal{P}$ is defined as: $\sum_{i=1}^{k} |p_i|$.

3.3 Formulating the Co-MAPF Problem

We wish to incorporate cooperative behavior into the classical MAPF problem. This is achieved by defining heterogeneous agents that collaborate and work together on a cooperative task. More specifically, we replace agent goals with a set of cooperative tasks, i.e., tasks that require the cooperation and coordination of a group of agents in order to be completed. Here we limit ourselves to cooperative tasks (simply referred to as tasks in the rest of this thesis) that require pre-defined pairs of agents to meet. We discuss possible extensions in Chapter 7.

3.3.1 Problem input

We are given a graph $G = (V, E)$, similar to the classical MAPF problem. In Co-MAPF, the set of agents A consists of two distinguishable sets, i.e., $A = \mathcal{A} \cup \mathcal{B}$. Each set includes k agents of a specific type, namely $\mathcal{A} = \{\alpha_1, \ldots, \alpha_k\}$ and $\mathcal{B} = \{\beta_1, \ldots, \beta_k\}$ (a total of $2k$ agents). The two types of agents may represent robots that differ in their traversal capabilities or possible actions in different locations (for instance, picking up an object).

Each agent has a unique start location given by a function $\mathcal{V}_0 : A \to V$ s.t.$\mathcal{V}_0(a)$ is the location of agent a at time step 0, namely $p_i[0] = \mathcal{V}_0(a_i)$.

In Co-MAPF, agent goals are not given explicitly. Instead, we are given a set of cooperative tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_k\}$ s.t. each task τ_i is assigned to a pair of agents (α_i, β_i). In the context of these tasks, we refer to α_i and β_i as the *initiator* and *executor* agents, respectively. Each task $\tau_i \in \mathcal{T}$ is defined by a start location s_i and a goal location g_i.

An agent's goal is then derived from its assigned task. The initiator agent initiates the task by arriving at the task's start location. The executor agent executes the task by arriving at the task's goal location. In between, both agent must meet for the task to be completed. More specifically, in our setting, a task $\tau_i = (s_i, g_i)$ for agents (α_i, β_i) is composed of the following steps:

i) Moving the initiator agent from its start location to the task start location. Namely, α_i from $\mathcal{V}_0(\alpha_i)$ to s_i.

ii) Moving the initiator agent from the task start location to a so-called *meeting* at a specific time. Namely, α_i from s_i to $m_i = (v_i^m, t_i^m)$ where $v_i^m \in V$ is the meeting location and t_i^m is the meeting time step, both of which are computed by the algorithm (and not specified by the task[1]).

iii) Moving the executor agent from its start location to the meeting location at the meeting time. Namely, β_i from $\mathcal{V}_0(\alpha_i)$ to v_i^m at time t_i^m.

iv) Moving the executor agent from the meeting location to the task goal location. Namely, β_i from v_i^m to g_i.

Following these steps, the initiator agent has to plan a path from its start location to the task start location and then to *meet* with the executor agent. The executor has to plan a path

[1]Note that a meeting m_i is defined by its location and time. Thus, when referring to a meeting m_i, we mean both it location v_i^m and time t_i^m.

from its start location to first meet the initiator and then complete the task by planning a path to the task's goal location. For a visualization, see Figure 3.1.

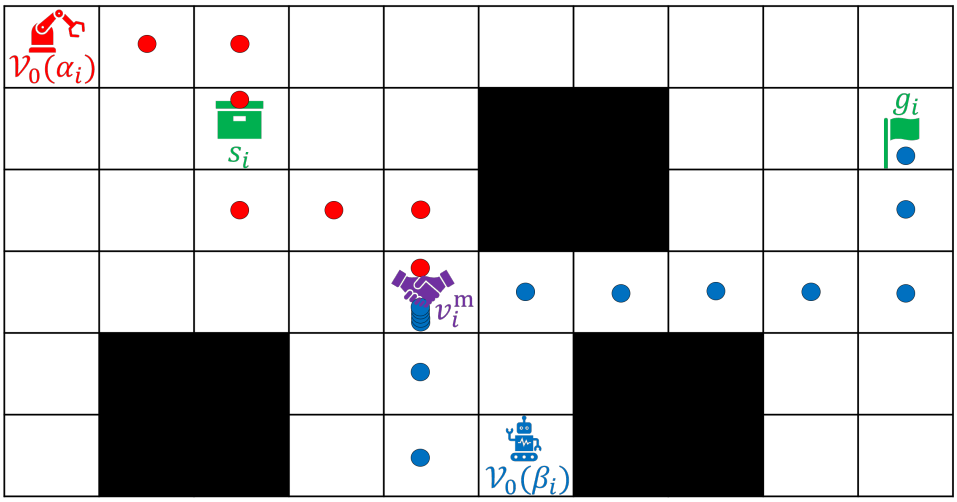


Figure 3.1: The paths completing a single cooperative task. The initiator agent (in red) plans a path from its start location ($\mathcal{V}_0(\alpha_i)$) to the task start location (s_i, depicted with a green box) and then to the meeting (v_i^m, depicted with a purple handshake). The executor agent (in blue) plans a path from its start location ($\mathcal{V}_0(\beta_i)$) to the meeting and then to the task goal location (g_i, depicted with a green flag). Note that in this example, the blue agent has to wait at the meeting location several time steps, until the designated meeting time.

Note that a meeting (v_i^m, t_i^m) is defined using a single vertex v_i^m, which means that both agents arrive at the same location at the same time, and this is not considered a collision in our setting. This represents a scenario where the two robots meet following some special *rendezvous* protocol (as depicted by robots #2 in Figure 1.2), allowing them to be at the same location without colliding. However, we may also define a meeting using a pair of adjacent vertices, i.e. $v_{i,\alpha}^m$ and $v_{i,\beta}^m$ where each agent has its own meeting location, similar to [Revach et al., 2020]. This extension is discussed in Chapter 7.

3.3.2 Co-MAPF solution

A Co-MAPF solution is a set of path pairs $\mathcal{P} = \left\{(p_1^\alpha, p_1^\beta), \ldots, (p_k^\alpha, p_k^\beta)\right\}$ such that for each pair $1 \leq i \leq k$, p_i^α, p_i^β start in $\mathcal{V}_0(\alpha_i)$ and $\mathcal{V}_0(\beta_i)$, respectively. Path p_i^α goes through s_i at some time step t_i, and both paths contain a meeting at vertex v_i^m at the same time t_i^m s.t. $t_i \leq t_i^m$. Finally, p_i^α ends in vertex v_i^m at time t_i^m and p_i^β ends in vertex g_i. Namely, for each $1 \leq i \leq k$,

$$p_i^\alpha = \left\{ \underbrace{\mathcal{V}_0(\alpha_i), \ldots, s_i, \ldots, v_i^m}_{t_i^m \text{ time steps}} \right\}, \tag{3.1}$$

$$p_i^\beta = \left\{ \underbrace{\mathcal{V}_0(\beta_i), \ldots, v_i^m}_{t_i^m \text{ time steps}}, \ldots, g_i \right\}. \tag{3.2}$$

Similarly to classical MAPF, in order for a solution to be feasible, there should be no conflicts between the paths in $\mathcal{P}$, with the exception that the paths of agents sharing a task intersect at

their meeting point.

An optimal Co-MAPF solution is a feasible set of path pairs $\mathcal{P}$ which optimizes some objective function (specifically defined in next paragraph).

3.3.3 Co-MAPF objective functions

Similar to classical MAPF, we define the makespan (MKSP) and sum-of-costs (SOC) objective functions in the cooperative case.

- The **makespan** of a Co-MAPF solution $\mathcal{P} = \left\{(p_1^\alpha, p_1^\beta), \ldots, (p_k^\alpha, p_k^\beta)\right\}$ is defined as $\max_{1 \le i \le k} |p_i^\beta|$. Note that we take maximum over paths of the executor agent only since $\forall i,\ |p_i^\beta| \ge |p_i^\alpha|$.
- The **sum-of-costs** of $\mathcal{P}$ is defined as $\sum_{1 \le i \le k} |p_i^\alpha| + |p_i^\beta|$. Wait actions are counted until an agent finishes its plan (i.e., after the meeting for α_i and after arriving at g_i for β_i).

In this work we focus on the SOC objective, which is, arguably, more natural for our setting—it implicitly minimizes both the time it takes to complete a task, and the time the initiator finishes its part in the task. We note that all results presented in this thesis can be applied to the MKSP objective as well.

3.4 Source-Connected Co-MAPF Instances

An important characteristic of a MAPF instance is whether it is solvable, i.e., there exists a feasible solution. Figure 3.2 shows a simple non-solvable MAPF instance with one agent. The agent starts at s_1 and has to arrive at g_1, however, there does not exist a path from s_1 to g_1.

Figure 3.2: A non-solvable MAPF instance.

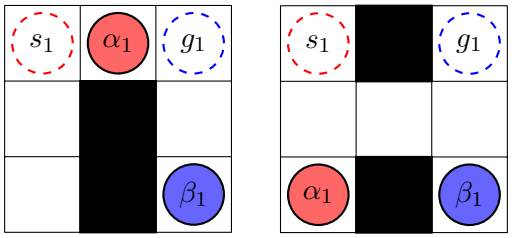
For classical MAPF, it is possible to check if an instance is solvable in polynomial time [Yu and Rus, 2014]. In the Co-MAPF setting, given a set of meeting locations, we may decompose the problem into two MAPF instances and check that both are solvable. However, Co-CBS also searches for different meeting locations (and times), which means that agents' (intermediate) goals are determined during the search and not pre-defined. Therefore, to be able to efficiently check if a Co-MAPF instance is solvable, we define the notion of *source-connected* instances. The intuition behind the source-connected definition is that agents can rest (that is, stay forever) in their start locations, such that they cannot block the execution of other tasks.

We denote a path from vertex u to vertex v in short by $u \to v$.

Definition 3.4.1 (source-connected instances)**.** A source-connected Co-MAPF instance is an instance in which for each task τ_i, the following paths exist: (i) $\mathcal{V}_0(\alpha_i) \to s_i$, (ii) $\mathcal{V}_0(\beta_i) \to s_i$, and (iii) $s_i \to g_i$, and none of them pass via an agent's start location.

While still being general, source-connected instances are always solvable, and we can efficiently check the condition in the above definition. Both these claims will be proved shortly.

We also state that Co-MAPF instances which are not source-connected, may still have a valid solution. Moreover, our suggested algorithm (presented in the next chapter) will also solve most non source-connected instances. Figure 3.3a illustrates these ideas.



(a) (b)

Figure 3.3: A not source-connected (a) and source-connected (b) Co-MAPF instances. Black squares are obstacles. In (a), the paths $s_1 \to g_1$ and $\mathcal{V}_0(\beta_1) \to s_1$ do not adhere to the condition in Definition 3.4.1, as they pass an agent's start location. In (b), all relevant paths do not pass an agent's start location. Nevertheless, both instances are solvable.

Claim 3.4.2. *Checking if a Co-MAPF instance is source-connected can be done in polynomial time in the graph size.*

Proof. Denote V_0 the set of all agents' start locations, namely $V_0 = \{\mathcal{V}_0(\alpha_i)\}_{i=1}^{k} \cup \{\mathcal{V}_0(\beta_i)\}_{i=1}^{k}$. Denote G' a sub graph of G that contains only vertices in $V \setminus V_0$. For each task $\tau_i \in \mathcal{T}$, given (s_i, g_i) and (α_i, β_i), we can calculate the following paths in G': $\mathcal{V}_0(\alpha_i) \to s_i$, $\mathcal{V}_0(\beta_i) \to s_i$, and $s_i \to g_i$. This can be done using any polynomial-time graph-search algorithm, such as A* or Dijkstra to test if these paths exist. Therefore, by performing this test for each task independently, we can determine that the instance is source-connected (if all paths exist in G') or conclude that it's not (if at least one of the paths does not exist) in polynomial time in the graph size. More specifically, by running Dijkstra's algorithm for each task τ_i (from source vertex s_i), we get that the complexity of checking the source-connected property is $\mathcal{O}(|\mathcal{T}| \cdot (|V|log|V| + |E|))$.■

Lemma 3.4.3. *Every source-connected Co-MAPF instance is solvable.*

Proof. Solving a source-connected Co-MAPF instance can be done by planning a solution for each pair of agents independently from the other agents, with a meeting at the task start location, at the earliest time possible for both agents. For each task $\tau_i \in \mathcal{T}$ it is guaranteed by the source-connected definition that the following paths exist and do not pass via other agents' start location:

i) For agent α_i, p_1 from $\mathcal{V}_0(\alpha_i)$ to s_i.

ii) For agent β_i, p_2 from $\mathcal{V}_0(\beta_i)$ to s_i, and p_3 from s_i to g_i.

We define the solution paths for this pair of agents to be: $p_i^{\alpha} = p_1$ and $p_i^{\beta} = p_2 \cdot p_3$ (where $(\cdot)$ denotes a concatenation of the paths which is well-defined for the mentioned paths). We also assume that the paths include wait actions at the starting points as necessary in order to guarantee that the other agent can traverse its path successfully. We also require α_i to return to its starting location after meeting with β_i, and β_i to return to its starting location after

completing the task. This ensures that p_j^α and p_j^β are clear paths for any other agents α_j, β_j so the constructed solution is feasible. ■

To summarize, in this chapter we described and formally defined the Co-MAPF problem, and introduced high-level agent interactions via meetings in the context of cooperative tasks. In the next chapter we describe the approach for solving the problem, as well as our suggested algorithm.

Chapter 4

Optimal Algorithm for Solving the Co-MAPF Problem

In the previous chapter we described the cooperative model for the Co-MAPF problem, i.e., scheduling a meeting between two agents working together on a cooperative task. In this chapter we discuss our approach for solving the Co-MAPF problem optimally, and present our suggested *Cooperative Conflict-Based Search (Co-CBS)* algorithm. We provide a theoretical analysis of Co-CBS and suggest two improvements to the basic algorithm.

To solve the Co-MAPF problem, besides finding collision-free shortest paths for agents, we also need to determine the set of meetings, one meeting (location and time) for each task. Meetings and tasks are coupled—once a meeting is set, agent paths are planned accordingly to and from the meetings (see Figure 3.1 for illustration). The main challenge is therefore to simultaneously search for meetings and paths, such that the total solution cost is minimized (see Section 3.3.3). Using a centralized A*-based approach has two problems. First, it is not straightforward to plan agents' high-level interactions (i.e., determine a meeting location and time for each task) since we need to coordinate agents' actions over time. We need to account for the total cost of completing a task given a meeting location and time, and an infinite number of goal states exist using such an approach. Second, a centralized planner would suffer from a run-time exponential blowup in the number of tasks, similar to the classical MAPF problem.

We may also consider a prioritized approach that would plan for each agent independently. However, we still need to determine meeting locations and times, and such a planner may potentially be very sub-optimal. Indeed, in Chapter 6 we observe this in an evaluation of such a prioritized planner compared to our suggested approach.

To deal with the aforementioned challenges, we wish to decouple the search for a meeting from the search of paths, and resolve conflicts in an efficient manner. We refer to the space of all possible meeting locations and times as the *meetings space* and note that it is infinite, since we can consider any time step to meet. Therefore, it is not possible to enumerate all sets of meetings (one meeting for each task), and we need an efficient way to generate sets of meetings to perform a best-first search in the meetings space.

In this chapter we present our suggested *Cooperative Conflict-Based Search (Co-CBS)* algorithm based on the aforementioned ideas to solve the Co-MAPF problem. Co-CBS is based on two previously-suggested algorithms: the well-known Conflict-Based Search (CBS) [Sharon

et al., 2015] for solving a classical MAPF problem and the Conflict-Based Search with Optimal Task Assignment (CBS-TA) [Hönig et al., 2018] for solving the anonymous MAPF problem, where we also need to assign goals (or tasks) to each agent. For clarity of exposition, we first describe CBS, followed by a detailed description of Co-CBS.

4.1 Conflict-Based Search (CBS)

CBS [Sharon et al., 2015] is a state-of-the-art optimal algorithm for solving MAPF instances. It is a two-level search algorithm, that implements two main ideas: planning for each agent independently, and resolving conflicts by imposing constraints on single-agent paths. The high-level performs a best-first search over a search tree called a *constraint tree* (CT). Each CT node consists of a solution, its cost and a set of constraints. A solution is a set of paths, one for each agent (see Section 3.2.4), from its start location to its goal location. Constraints is a set of constraints for each agent. CBS finds conflicts in the solution and resolves them by imposing constraints on agents. More specifically, CBS defines two types of constraints, corresponding to the two types of conflicts (defined in Section 3.2.3):

i) A *vertex constraint* (a, v, t), forbidding agent a to be at location v at time step t.

ii) An *edge constraint* (a, u, v, t), forbidding agent a to cross edge (u, v) at time step t.

The low-level search constructs paths for each individual agent while satisfying the imposed constraints. CBS resolves conflicts by splitting a CT node and introducing an additional constraint for each agent participating in the conflict at the lower level.

The search starts with a root node with an empty set of constraints. Paths are planned for each agent independently, the solution is calculated and the root CT node is then inserted into the OPEN list. At each iteration, a lowest-cost node from the OPEN list is selected for expansion. During the expansion process, CBS looks for conflicts in the node's solution. Once a conflict is found, the CT is split, creating two new CT nodes, with a constraint added to each, corresponding to the two agents involved in the conflict. Paths are planned for each CT node, for the corresponding agent and while satisfying all the constraints. The cost of the newly computed solution is calculated, and the new CT nodes are then inserted into the OPEN list. The search is over when CBS expands a CT node from the OPEN list and finds that it has no conflicts in its solution. It is then returned as the optimal solution. CBS is optimal and complete. Full details can be found in [Sharon et al., 2015].

4.2 Cooperative Conflict-Based Search (Co-CBS)

We now continue with an overview of Co-CBS followed with lower-level details. Co-CBS is outlined in Algorithm 4.1, with the main changes from CBS highlighted. An execution example is described in Section 4.1 and depicted in Figure 4.2.

Algorithm 4.1 Cooperative Conflict-Based Search (Co-CBS)

Input: $G, \mathcal{A}, \mathcal{B}, \mathcal{V}_0, \mathcal{T}$ ▷ Co-MAPF problem instance, see Section 3.3.1
Returns: optimal path for each agent
for all $\tau_i \in \mathcal{T}$ **do** ▷ using Algorithm 4.2 for each task
    $T_i \leftarrow compute\_meetings\_table(\tau_i, \alpha_i, \beta_i)$
R = new node
$R.constraints \leftarrow \emptyset$
$R.meetings \leftarrow$ get the initial set of minimal-cost set of meetings $\mathcal{M}^*$
$R.root \leftarrow$ True
$R.solution \leftarrow plan\_paths()$ ▷ to and from meetings
$R.cost \leftarrow compute\_cost(R.solution)$
insert R to OpenRoots
while Open *not empty* or OpenRoots *not empty* **do**
    $N \leftarrow$ lowest cost node from Open∪OpenRoots ▷ pop a node from the Open list
    Validate the path in N until a conflict occurs
    if N has no conflicts **then**
        return $N.solution$ ▷ N is goal
    if $N.root$ *is* $True$ **then**
        $expand\_root(N)$ ▷ using Algorithm 4.3
    $C \leftarrow$ first conflict (a_i, a_j, v, t) in N
    for all *agent* a_i *in* C **do**
        $A \leftarrow$ new node
        $A.constraints \leftarrow N.constraints + (a_i, v, t)$
        $A.meetings \leftarrow N.meetings$
        $A.root \leftarrow$ False
        $A.solution \leftarrow N.solution$
        Update $A.solution$ by invoking $plan\_paths(a_i)$
        $A.cost \leftarrow compute\_cost(A.solution)$
        Insert A to Open

4.2.1 Algorithm Overview

Co-CBS is a search algorithm based on CBS that considers the cooperative aspect of the problem. More specifically, Co-CBS consists of three levels of search in three different spaces (similar to [Hönig et al., 2018] and [Surynek, 2020]): (i) the *meetings space*, (ii) the *conflicts space* and (iii) the *paths space*. The meetings space contains all possible combinations of meetings, one for each task. We'll refer to the three levels of search as the meetings level, conflicts level and paths level, respectively.

Co-CBS simultaneously searches over all possible meetings and for each meeting, over all possible paths. To perform this search in a systematic and efficient manner, we need to consider an *ordering* of the meetings. Indeed, in Equation 4.1 we define a meeting's cost which is dependent both on the meeting's location and time. To efficiently traverse the set of possible meetings, we introduce the notion of a *Meetings Table* which stores for each meeting location the currently-best meeting time. As we will see, this table will allow us to iterate over all meetings in a best-first manner.

In contrast to CBS that constructs a single constraint tree (CT), Co-CBS creates a forest of CTs, similar to [Hönig et al., 2018]. Each CT starts in a *root* node and corresponds to a

specific set of meetings (a specific meeting for each task). In Co-CBS, each CT node has two additional fields (when compared to CBS): *root* specifies if the node is a root or a *regular* node and *meetings* specifies the current set of meetings (one for each task) which is used during the path-level search.

Co-CBS starts with a single root node, with the minimum-cost set of meetings (see Equation 4.4), while ignoring possible conflicts between agents. In each iteration, Co-CBS selects a lowest-cost node from the OPEN list (either a root or regular node), in a best-first approach similar to CBS. Whenever a root node is selected for expansion, in addition to splitting the tree due to a conflict in its solution, Co-CBS also expands it in the meetings space by generating the next best sets of meetings. Namely, new root nodes are created only on demand. For each expanded node, given its set of meetings and constraints, the paths level computes a solution by planning the different steps a task solution is composed of (see Section 3.3.1).

4.2.2 Computing the Meetings Table

We denote the cost of a meeting $m_i = (v_i^m, t_i^m)$ by $C_i(m_i) \triangleq C_i(v_i^m, t_i^m)$. C_i is given for the SOC objective, by

$$C_i(v,t) = \begin{cases} 2 \cdot t + d\left(v, g_i\right), & t \geq t_i^*\left(v\right) \\ \infty, & \text{otherwise} \end{cases}, \tag{4.1}$$

where $t_i^*\left(v\right)$ is the earliest possible meeting time at v for task τ_i, i.e., the earliest time both assigned agents can arrive at v. Specifically, $t_i^*\left(v\right)$ is defined as

$$t_i^*\left(v\right) = \max\left\{d\left(\mathcal{V}_0\left(\alpha_i\right), s_i\right) + d\left(s_i, v\right), d\left(\mathcal{V}_0\left(\beta_i\right), v\right)\right\}, \tag{4.2}$$

where $d(u,v)$ is the length of the single-agent shortest path from u to v. If $d(u,v) = \infty$, no such path exists.

The first step of Co-CBS is to compute T_i, the meetings table for each task τ_i (lines 3-4 in Algorithm 4.1). The meetings table is a function $T_i : V \to \mathbb{R} \cup \{\infty\}$ that returns for each vertex $v \in V$ the cost of completing task τ_i with a meeting in v at the currently-best time. $T_i\left(v\right)$ is initialized for each $v \in V$ with $T_i\left(v\right) = C_i(v, t_i^*\left(v\right))$, namely the earliest time possible for each location. Each meetings table is stored as a heap which allows for `update` and `getMin` operations in $\mathcal{O}(\log|V|)$. These operations are used during the root node expansion which will be described shortly.

We compute $T_i\left(v\right)$ for all $v \in V$ in polynomial time (in the size of the graph) using A* and Dijkstra's algorithm as described in Algorithm 4.2. For each task τ_i, we compute for every node $v \in V$ the paths to the agents' start locations $\left(\mathcal{V}_0\left(\alpha_i\right), \mathcal{V}_0\left(\beta_i\right)\right)$, as well as task's start (s_i) and goal (g_i) locations. This is done by running Dijkstra's algorithm three times (from source vertices $s_i, g_i, \mathcal{V}_0\left(\beta_i\right)$). Then, for every node $v \in V$, we compute $t_i^*\left(v\right)$ the earliest possible time meeting in v (using equation 4.2), and $C_i(v, t_i^*\left(v\right))$ the cost of meeting at v at the earliest time possible (using equation 4.1). All these values are stored in the table for later use.

Algorithm 4.2 Compute Meetings Table

1: **Input:** A Task τ_i and two assigned agents α_i and β_i
2: **Returns:** T_i, the meetings table for task τ_i
3: Compute $d\left(\mathcal{V}_0\left(\alpha_i\right), s_i\right)$ ▷ initiator start to task start
4: Compute $d\left(s_i, v\right), \forall v \in V$ ▷ task start to all vertices
5: Compute $d\left(\mathcal{V}_0\left(\beta_i\right), v\right), \forall v \in V$ ▷ executor start to all vertices
6: Compute $d\left(v, g_i\right), \forall v \in V$ ▷ task goal to all vertices
7: **for all** $v \in V$ **do**
8: Calculate $t_i^*\left(v\right)$ ▷ earliest meeting time in v
9: Calculate $T_i\left(v\right)$ ▷ task cost with meeting at v
10: Store $\left(v, t_i^*\left(v\right), T_i\left(v\right)\right)$ in table

4.2.3 Root Initialization

We define the cost of a set of meetings $\mathcal{M} = \{m_1, \ldots, m_k\}$ as follows:

$$C\left(\mathcal{M}\right) = \sum_{i=1}^{k} C_i\left(v_i^m, t_i^m\right). \tag{4.3}$$

$\mathcal{M}^*$ is a set of meetings that minimizes the problem objective while ignoring possible conflicts between agents. Namely,

$$\mathcal{M}^* \in \arg\min_{\mathcal{M}} C\left(\mathcal{M}\right). \tag{4.4}$$

Co-CBS's search starts with creating the initial CT root node with an empty set of constraints, and a minimal-cost set of meetings $\mathcal{M}^*$, by choosing a lowest-cost meeting for each task from the meeting tables (lines 5-8). Given $\mathcal{M}^*$, the paths level is called to compute individual paths for each agent (line 9). This is similar to CBS, except that in the path-level search we plan for each task τ_i in parts: (i) for α_i from $\mathcal{V}_0\left(\alpha_i\right)$ to s_i, and then from s_i to v_i^m at time t_i^m, and (ii) for β_i from $\mathcal{V}_0\left(\beta_i\right)$ to v_i^m at time t_i^m and then to g_i. Note that when planning for a meeting, we should consider both the meeting location and time. The initial CT root node cost is computed and it is inserted to the OPEN list (lines 10-11).

4.2.4 Node Selection

As long as there are nodes in the OPEN list (line 12), we follow CBS's best-first search approach and select a node with a lowest cost (line 13). If the OPEN list contains both root and regular nodes with the same lowest cost, Co-CBS chooses to expand a regular node (to perform this in practice, Co-CBS keeps root and regular nodes in two separate OPEN lists).

4.2.5 Root Node Expansion

After selecting a lowest-cost node N from the OPEN list, Co-CBS checks for conflicts in its solution (line 14). If none are found, $N.solution$ is returned as the optimal solution (lines 15-16). Otherwise, if N is a root node, it is expanded to get its successors in the meetings space (lines 17-18). The process of expanding a root node is described in Algorithm 4.3. Given the current set of meetings (in the expanded root node) $\mathcal{M} = \{m_1, \ldots, m_k\}$, Co-CBS generates up to k new sets of meetings, one for each task. This is done in a non-decreasing manner, by

replacing one meeting $m_i \in \mathcal{M}$ at a time, an idea similar to the Increasing Cost Tree Search (ICTS) [Sharon et al., 2013] algorithm, thus creating k new root nodes.

Algorithm 4.3 Expand root

1: **Input:** Meetings tables of all tasks, a root node P
2: **for all** $\tau_i \in \mathcal{T}$ **do** ▷ loop over all tasks
3: R = new node
4: $R.constraints = \emptyset$
5: $R.meetings = P.meetings$
6: $R.meetings[\tau_i] = get\_next\_meeting(T_i)$ ▷ change only τ_i's meeting
7: $R.root$ = True
8: Update $R.solution$ by invoking $plan\_paths(\alpha_i, \beta_i)$
9: $R.cost = compute\_cost(R.solution)$
10: insert R to OpenRoots

For each newly created root node, we replace only one meeting of one task, by selecting the next-best meeting for this task. This process is described in Algorithm 4.4. To get the next-best meeting for task τ_i, we have to search both for different locations and time steps in the meetings space. The meetings table T_i of τ_i initially consists of meetings at each possible location, at the earliest time possible. Each time Co-CBS invokes the `get-next-meeting` procedure for τ_i (line 6 in Algorithm 4.3), it returns the lowest-cost meeting $m_i = (v_i^m, t_i^m)$ from T_i. The table is then updated so that it holds the next lowest-cost meeting. This is done by updating $T_i(v_i^m) = C_i(v_i^m, t_i^m + 1)$. Namely, updating the cost of meeting at v_i^m, but at time $t_i^m + 1$ rather than t_i^m. The next time the `get-next-meeting` procedure is invoked, the next best meeting will be returned by the table.

Algorithm 4.4 Get next meeting

1: **Input:** Meetings table T_i, current meeting of task τ_i, (v_i^m, t_i^m)
2: **Returns:** The next-best meeting for task τ_i
3: $t_i^m(v_i^m) = t_i^m(v_i^m) + 1$ ▷ earliest meeting time in v_i^m
4: Update $T_i(v)$ ▷ set the new cost of meeting at v, namely at time $t_i^m + 1$
5: Return the lowest cost meeting in T_i

Subsequently, a new path is planned for the pair of agents whose meeting changed, the new CT node cost is computed and it is inserted into the Open list.

4.2.6 Conflicts Resolution

The last part of the algorithm is almost identical to CBS: when expanding a node N (either root or regular) Co-CBS splits its CT and creates a regular node for each agent by the first conflict found (lines 19-28). These nodes has the same set of meetings as N (line 23).

4.3 Co-CBS Example

We now demonstrate the execution of Co-CBS on a problem instance, depicted in Figure 4.1. In this problem we have two tasks, namely τ_1 and τ_2. Agents α_1 and β_1 execute τ_1 from s_1 to g_1 and agents α_2 and β_2 execute τ_2 from s_2 to g_2 (see Figure 4.1a).

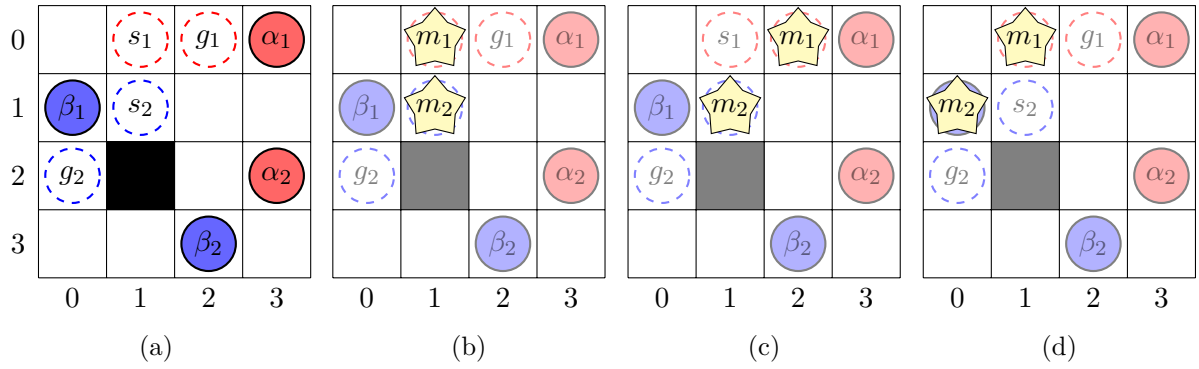


Figure 4.1: A Co-MAPF problem instance (a) with two tasks, and three different sets of meetings (one meeting for each task): $\mathcal{M}_1$ (b), $\mathcal{M}_2$ (c) and $\mathcal{M}_3$ (d). Agents α_1 and β_1 execute task τ_1 from s_1 to g_1 and agents α_2 and β_2 execute task τ_2 from s_2 to g_2.

Figure 4.2 shows the forest of constraint trees constructed during the execution of Co-CBS of the instance depicted in Figure 4.1a. Root and regular nodes are denoted by R and N, respectively.

Co-CBS start by creating the initial root node R_1, that contains the lowest-cost set of meetings (see Equation 4.4) $\mathcal{M}_1 \triangleq \mathcal{M}^*$. More specifically, $\mathcal{M}_1$ contains a meeting in $(1, 0)$ at time 2 for task τ_1 and a meeting in $(1, 1)$ at time 3 for task τ_2, as shown in Figure 4.1b. Paths are then planned for all agents via these meetings, with cost 13, and R_1 is inserted into the OPEN list.

In the first iteration, R_1 is extracted from the OPEN list, as it is the only node. Co-CBS validates its solution and finds a conflict at time $t = 1$. The CT is split into two new regular nodes N_1, N_2 with the corresponding constraints. Paths are planned for both nodes, resulting in cost of 13 for N_1, and no solution for N_2. N_1 is inserted into the OPEN list, while N_2 is discarded. In addition, since R_1 is a root node, it is also expanded to new root nodes R_2 and R_3, with meeting sets $\mathcal{M}_2, \mathcal{M}_3$, shown in Figures 4.1c and 4.1d, respectively. Note that only one meeting is changed at a time (compared to the parent root node R_1)—m_1 in R_2 and m_2 in R_3. Paths are planned for both nodes, resulting in a cost of 14 for both, which are then inserted into the OPEN list.

Co-CBS continues to expand nodes in a best-first manner: N_1 is chosen next for expansion, creating N_3, N_4, both with no solution. In the next iterations, Co-CBS will expand R_2 (to R_4, R_5, N_5, N_6), then N_5, and finally R_3. Since R_3 has no conflicts, its solution will be returned as a feasible optimal solution, and the search is over.

4.4 Co-CBS Theoretical Analysis

We now provide a theoretical analysis of Co-CBS. More specifically, we discuss the conditions under which Co-CBS is complete, and prove that it's optimal on any solvable Co-MAPF instance.

4.4.1 Co-CBS Completeness

As discussed in Section 3.4, it is not possible to efficiently check whether a Co-MAPF instance is solvable by decomposing it to classical MAPF instances. This is since agents' goals are not predefined and may change during the search for meeting locations. We therefore restrict our dis-

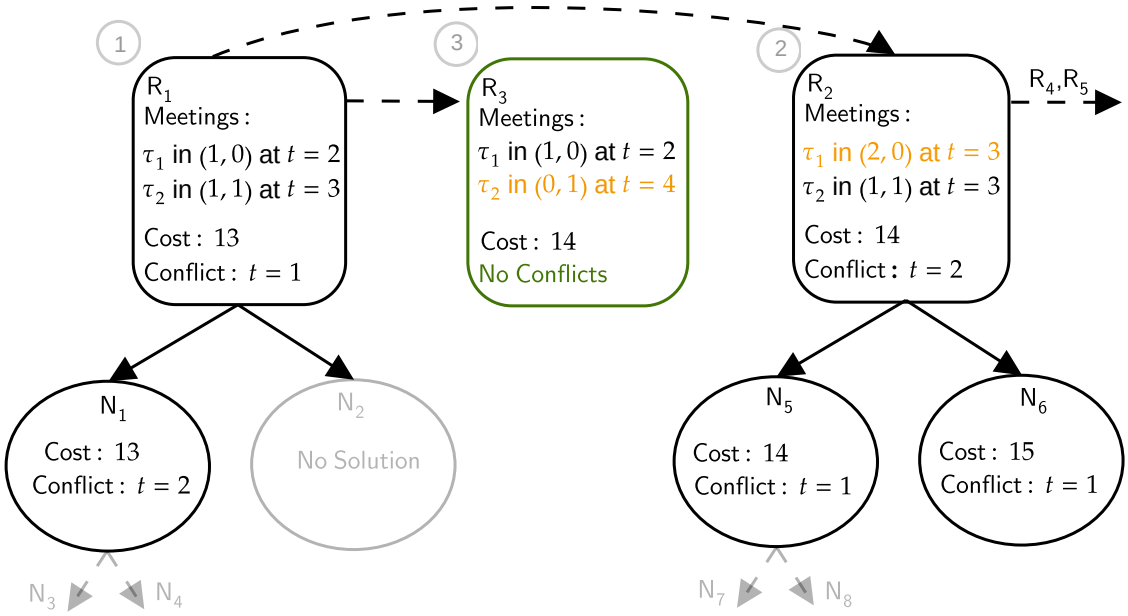


Figure 4.2: The search forest constructed during the execution of Co-CBS on the instance depicted in Figure 4.1a.

cussion of Co-CBS completeness to source-connected Co-MAPF instances (see Definition 3.4.1). By using the notion of source-connected Co-MAPF instances, we are able to guarantee Co-CBS's completeness while still solving a very wide and realistic set of problem instances. Investigating Co-CBS's completeness in the more general case is left for future research. For simplicity, we also assume similar to [Ma et al., 2019], a *disappear-at-target* behavior [Stern et al., 2019]. More specifically, the initiator agent disappears after the meeting, and the executor agent disappears after completing the task (at the task goal location). Note that the following proofs still work without this assumption. A more interesting scenario is where agents are assigned with new tasks upon finishing their part, commonly known as the *lifelong planning* problem as discussed in Chapter 7.

We wish to show that Co-CBS performs a best-first exhaustive search in the meetings space, by generating sets of meetings with non-decreasing cost every time a root node is expanded. We start by showing that the `get-next-meeting` procedure, described in Algorithm 4.4, generates meetings (for a single task) with non-decreasing costs.

Lemma 4.4.1. *The `get-next-meeting` procedure is exhaustively generating meetings of non-decreasing-cost.*

Proof. Consider T_i, the meetings table of task τ_i which is assigned to agents (α_i, β_i). Denote n as the number of times `get-next-meeting` has been invoked. For $n = 1$, because of how T_i is initialized, `get-next-meeting` returns the meeting location and time with lowest cost. Assume by induction, that the n'th time `get-next-meeting` is invoked, it returns the n'th best meeting in the location-time space. Denote this meeting location and time $\hat{v}_n$ and $\hat{t}_n$, respectively. Denote the total cost of the paths of α_i and β_i via this meeting $\hat{c}_n$. After the n'th invocation, T_i is updated such that the meeting at $\hat{v}_n$ is at time $(\hat{t}_n + 1)$ and costs $(\hat{c}_n + 2)$ (for the SOC objective, since we add a time step for each agent). T_i is then sorted by meeting costs. This

means that in the $(n+1)$'th invocation of `get-next-meeting`, it will return the $(n+1)$'th best meeting (with cost $\leq (\hat{c}_n + 2)$). ■

We now show in the following lemma, that Co-CBS performs a best-first exhaustive search in the meetings space.

Lemma 4.4.2. *By expanding root nodes, Co-CBS is performing an exhaustive best-first search in the meetings space.*

Proof. We wish to prove that Co-CBS will eventually generate and examine all possible sets of meetings, and that it will do so in a best-first approach. To do so we consider a general set of meetings, and show that the number of sets of meetings generated before it is bounded. We also show that all sets of meetings generated before our general set has cost lower or equal.

Denote $m_i^* \triangleq m_i^0$ the lowest-cost meeting for task τ_i, m_i^1 the next best meeting, then m_i^2, and so on. Namely,

$$C_i(m_i^0) \leq C_i(m_i^1) \leq C_i(m_i^2) \cdots . \tag{4.5}$$

Let $\mathcal{M}' = \left\{m_1^{j_1}, \ldots, m_k^{j_k}\right\}$ be a set of meetings, where $m_i^{j_i}$ is the j_i'th best meeting for task τ_i. We define the *level* of $\mathcal{M}'$ as

$$\ell\left(\mathcal{M}'\right) = \sum_{i=1}^{k} j_i. \tag{4.6}$$

Note that due to (4.5), a set of meetings with a higher level have a higher or equal cost, namely

$$\ell\left(\mathcal{M}'\right) < \ell\left(\mathcal{M}''\right) \Rightarrow C(\mathcal{M}') \leq C(\mathcal{M}''). \tag{4.7}$$

Co-CBS starts by creating the initial root node R_0, with the lowest-cost set of meetings $\mathcal{M}^* \triangleq \mathcal{M}_0$, where $\mathcal{M}_0$ contains the lowest-cost meeting for each task. Namely,

$$\mathcal{M}_0 = \left\{m_1^0, \ldots, m_k^0\right\}.$$

Note that $\ell\left(\mathcal{M}_0\right) = 0$.

When R_0 is expanded, Co-CBS creates k new root nodes with k new sets of meetings, by replacing one meeting in each new root node, with the next best meeting. Namely, R_0 is expanded to root nodes with sets of meetings

$$\left\{\left\{m_1^1, m_2^0 \ldots, m_k^0\right\}, \left\{m_1^0, m_2^1 \ldots, m_k^0\right\} \ldots, \left\{m_1^0, m_2^0 \ldots, m_k^1\right\}\right\}.$$

In other words, Co-CBS expands all root nodes with sets of meetings that have level$= 1$. Generally speaking, when Co-CBS expands a root node with set of meetings with level ℓ', it creates up to k new root nodes with sets of meeting with level $\ell' + 1$ (note that some of the sets of meetings with level $\ell' + 1$ have already been created during a previous expansion). Furthermore, there are exactly $\binom{\ell'+k-1}{\ell'}$ different sets of meeting with level ℓ'.

Now, consider a specific set of meetings $\mathcal{M}'$ with cost $C(\mathcal{M}')$ and level $\ell\left(\mathcal{M}'\right)$. Denote by ℓ_h the highest level of sets of meetings with cost $\leq C(\mathcal{M}')$. Namely, all sets of meeting with

level $> \ell_h$ have cost $> C(\mathcal{M}')$. We know that there are exactly

$$\sum_{l=0}^{\ell_h} \binom{l+k-1}{l}$$

sets of meetings with level $\leq \ell_h$, and therefore *at most* that number of root nodes with cost $\leq C(\mathcal{M}')$. Since Co-CBS always selects a lowest-cost node for expansion, the number of root node expansions before creating root node with $\mathcal{M}'$ is bounded by $\sum_{l=0}^{\ell_h} \binom{l+k-1}{l}$. This means that the root node with sets of meeting $\mathcal{M}'$ will eventually be created, after creating all possible sets of meetings with lower cost. ■

We now continue to prove that Co-CBS is complete on source-connected instances (which are always solvable).

Theorem 4.1 (Co-CBS completeness)**.** *Co-CBS will return a solution for any source-connected Co-MAPF instance.*

Proof. Given a source-connected Co-MAPF, by Lemma 3.4.3 we know that there exists a solution for this instance. Denote the set of meetings in the solution by $\mathcal{M} = \{(v_1^m, t_1^m), \ldots, (v_k^m, t_k^m)\}$. From Lemma 4.4.2 we know that during the search, Co-CBS will create a root node, denoted by $R_{\mathcal{M}}$, whose set of meetings is $\mathcal{M}$. There exists a feasible solution such that each pair of agents (α_i, β_i) meet at (v_i^m, t_i^m). By the completeness of CBS it is guaranteed that the search from the CT root node $R_{\mathcal{M}}$ will eventually find the solution. ■

We've shown that Co-CBS is complete for source-connected Co-MAPF instances. We state however, that Co-CBS will also solve most instances where the source-connected property doesn't hold, albeit without a completeness guarantee.

4.4.2 Co-CBS Optimality

We show that Co-CBS returns an optimal solution for every *solvable* Co-MAPF instance, starting with the following lemma.

Lemma 4.4.3. *Let $\mathcal{M}$ be a set of meetings with cost $C(\mathcal{M}) = c$ and let N be a CT node with cost larger than c. Co-CBS will generate a root node corresponding to $\mathcal{M}$ before expanding N.*

Proof. Assume that there exists a set of meetings $\mathcal{M}$ s.t. $C(\mathcal{M}) = c$, that hasn't been generated yet. Assume by contradiction that Co-CBS expands a node N with a solution cost $c' > c$. By definition, the first generated set of meetings $\mathcal{M}^*$ (line 7 in Algorithm 4.1) induces a solution which minimizes the SOC objective function. This implies that the cost of completing all tasks in the (possibly infeasible) solution induced by $\mathcal{M}^*$ is less than or equal to c. The cost of completing all tasks in the (possibly infeasible) solution induced by $\mathcal{M}$ is equal to c. From Lemma 4.4.2 we know that each set of meetings generated between $\mathcal{M}^*$ and $\mathcal{M}$ has a cost smaller or equal to c. Furthermore, there must be at least one root node in the OPEN list consisting of one of these meeting sets. Therefore, there exists a root node that hasn't been expanded yet in the OPEN list with a cost smaller than c', in contradiction to Co-CBS's best-first search approach which chose node N with a larger cost for expansion. ■

Theorem 4.2 (Co-CBS optimality)**.** *Co-CBS returns an optimal solution for any solvable Co-MAPF instance.*

Proof. Assume that there exists an optimal solution with some cost c^*. Co-CBS performs a CBS-like search on each generated CT, namely, it searches through a forest of constraint trees. By Lemma 4.4.3 we get that the cost of each expanded root node of each CT constitutes a lower-bound on c^*. From the optimality guarantees of CBS, we get that any node expanded in each of those CTs (i.e., regular nodes) is also a lower bound on c^*. Due to Co-CBS's best-first approach, it won't expand a node with a cost larger than c^* before completing a search through all possible CT nodes with cost c^* (by expanding neither a root node nor a regular one). Since there exists a solution with such cost, and the number of possible solutions with a specific cost is finite, Co-CBS will eventually expand a node with an optimal and feasible solution and return it. ■

4.5 Improved Co-CBS: Prioritizing Conflicts and Lazy Expansion

In Section 4.2, we introduced the basic version of Co-CBS for solving the Co-MAPF problem optimally. Generally speaking, Co-CBS creates a forest of constraint trees and runs CBS on each tree. Therefore, we can apply previously-suggested CBS improvements to Co-CBS. One such improvement that has been shown to significantly decrease CBS's run-time is *prioritizing conflicts (PC)* [Boyarski et al., 2015b]. In this section we present in detail the application of PC to Co-CBS. More CBS improvements are discussed in Chapter 7. In addition, we introduce a unique improvement for Co-CBS called *Lazy Expansion (LE)*, which exploits special characteristics of root nodes. Both improvements keep Co-CBS optimal, while introducing a significant improvement in run time, as shown empirically in Chapter 6.

4.5.1 Prioritizing Conflicts (PC)

The Improved CBS (ICBS) algorithm [Boyarski et al., 2015b] introduced an enhancement to CBS by defining rules dictating how to split the CT when encountering a conflict in the solution. In particular, conflicts are divided into three types: *cardinal*, *semi-cardinal* and *non-cardinal*:

i) Cardinal conflict always causes an increase in the solution cost when adding its constraints to *any* of the agents involved in the conflict.

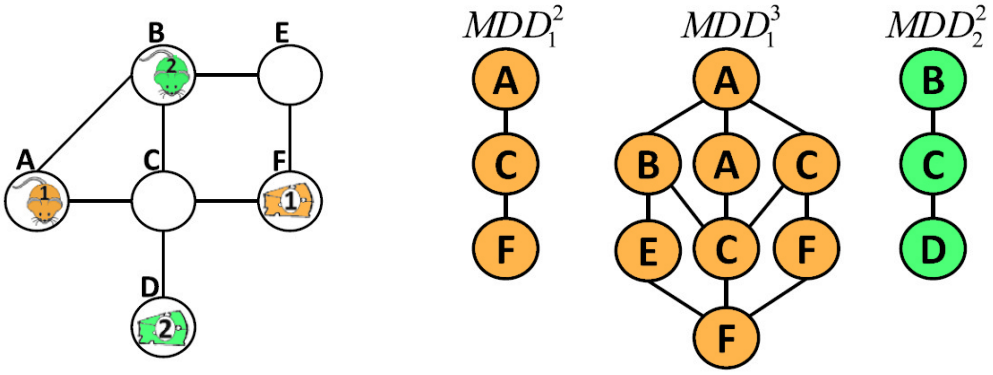
ii) Semi-cardinal conflict causes an increase in the solution cost when adding its constraints to *one* of the agents, leaving the cost of the second agent unchanged.

iii) Non-cardinal conflict does not cause an increase in the cost of neither agents.

When validating a solution of an expanded CT node, ICBS finds all conflicts in the solution and classifies each conflict to one of the three types. ICBS chooses to split cardinal conflicts first. If none exist, it chooses a semi-cardinal conflict, and finally a non-cardinal conflict. Cardinal conflicts are identified by examining the width of a *multi-value decision diagram (MDD)* [Sharon

et al., 2013], which is constructed for each low-level path found. The MDD is a directed a-cyclic graph which compactly stores all possible paths of a given cost c for a given agent, from its start vertex to its goal vertex. An MDD of cost c consists of c layers, corresponding to c time steps. The MDD of agent a of cost C is denoted MDD_a^c.

Figure 4.3 shows an example MAPF environment [Boyarski et al., 2015b] with two agents, and three MDDs: i) MDD_1^2 is the MDD of agent 1 of cost 2, ii) MDD_1^3 is the MDD of agent 1 of cost 3, and iii) MDD_2^2 is the MDD of agent 2 of cost 2.



(a) A MAPF environment.

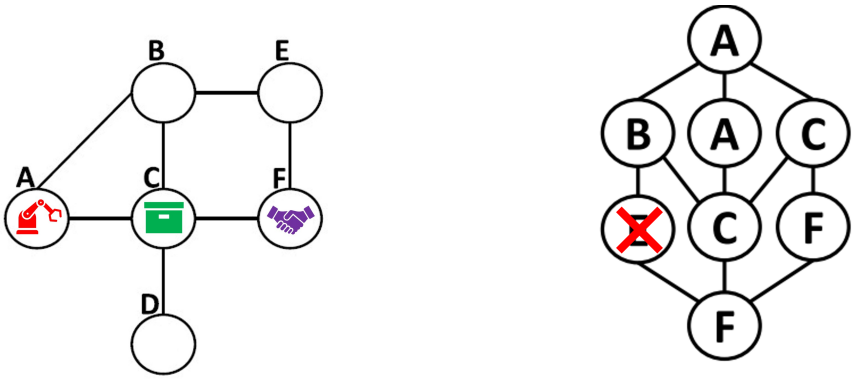
(b) Multi-valued decision diagrams (MDDs).

Figure 4.3: An example MAPF environment with two agents (a) and several corresponding MDDs (b).

Applying PC to Co-CBS is not straightforward, since an MDD stores paths from a start vertex to a goal vertex, while in Co-MAPF paths are constrained to ensure cooperation between agents. More specifically, in our Co-MAPF setting, each valid path has two phases. A path either traverses the task start location on its way to the meeting (for the initiator agent), or it traverses the meeting location (at a specific time) on its way to the goal (for the executor agent). We therefore need to modify the way an MDD is constructed, and indeed we suggest a method for efficiently doing so for both agents.

For the initiator agent, we must ensure it passes through the task's start location. In other words, we need to prune MDD nodes that are not part of any of the agent's paths which pass the task's start location. We refer to such nodes as *invalid nodes*. Constructing an MDD efficiently is done using two breadth-first searches–one forward and one backward (start to goal and vise versa) [Sharon et al., 2013]. In order to efficiently prune invalid nodes, we follow the following procedure: during the forward search, we mark MDD nodes corresponding to the task start location and all their descendants as *valid_forward*. Similarly, during the backward search, we mark these nodes and all their ancestors as *valid_backward*. Finally, all MDD nodes that are not marked with either flags are invalid and therefore pruned.

For a visualization, consider the Co-MAPF instance depicted in Figure 4.4a, with only one initiator agent. The meeting is at location F at time step 3, and the agent has to pass via the task start location at C on its way to the meeting. Figure 4.4b shows it corresponding MDD (of cost 3). Location E at time step 2 is invalid and is therefore pruned. Notice that being at location E at time step 2 makes it impossible for the agent to arrive at its meeting after passing the task start location.



(a) A Co-MAPF environment.

(b) Multi-valued decision diagram (MDD) with one invalid node.

Figure 4.4: An example Co-MAPF environment with one initiator agent (a) and its MDD of cost 3. (b).

For the executor agent, constructing the MDD requires only slight changes. We need to constrain the agent to be at the meeting's location at the meeting's time. We simply do it by eliminating all other nodes from the MDD layer corresponds to the meeting time during the forward pass in the MDD construction.

4.5.2 Lazy Expansion (LE) of Root Nodes

Co-CBS searches the meetings space by creating root nodes, each corresponding to a unique set of meetings. Note that since no constraints are imposed on paths of root nodes, their cost is given as an aggregation of their meeting costs. Namely, for each root node R_i, with a set of meetings $\mathcal{M}_i$,

$$R_i.cost = C(\mathcal{M}_i). \tag{4.8}$$

Furthermore, meeting costs are computed a-priori during the construction of meeting tables (see Section 4.2.2). This means that when a root node is expanded, and new root nodes are created, they can immediately be inserted into the OPEN list *without* computing their low-level paths, which is the most time-consuming step during the expansion process. The low-level paths will be computed only when these root nodes are extracted from the OPEN list. We term this *Lazy Expansion (LE)* of root nodes.

Each time a root node is expanded, it creates k new root nodes by replacing the meeting of each of the tasks. We emphasize that while generating those nodes is mandatory in order to guarantee optimality, most of them won't be expanded. Thus, the run-time saved by LE can be significant.

Chapter 5

Task Assignment for Cooperative MAPF

In our suggested Co-MAPF formulation, presented in Chapter 3, we assumed that tasks are pre-assigned to agents. Namely, each task $\tau_i \in \mathcal{T}$ is assigned to a pair of agents (α_i, β_i), and this is a part of the input to the Co-MAPF problem. However, since the solution cost is measured in total time steps to complete all tasks, determining which agents execute which task may have a dramatic impact on the cost. Therefore, in this chapter we relax this assumption and address the *Task Assignment (TA)* problem in the context of the Co-MAPF setting.

In the classical MAPF framework, the *Task Assignment and Path Finding (TAPF)* problem deals with assigning agents to goals, as well as finding conflict-free paths to agents. This problem is also known as the anonymous MAPF problem and can be solved in polynomial time for the makespan objective [Yu and LaValle, 2012]. For the sum-of-costs objective, the *Conflict-Based Search with Task Assignment (CBS-TA)* [Hönig et al., 2018] has been previously suggested. CBS-TA extends CBS by simultaneously searching over all possible assignments. The task assignment problem in the MAPF context is two dimensional. Informally speaking, given a $k \times k$ cost matrix $C = (c_{ij})$, we have to select k elements of C, so that there is exactly one element in each row and one in each column, and the sum of corresponding costs is minimized.

The task assignment problem in the Co-MAPF context is three dimensional: for each task τ_i we need to assign a specific initiator agent as well as an executor agent. This problem is equivalent to the *Multi-Index Assignment Problem (MIAP)* problem, and more specifically the axial 3-index assignment problem, which is NP-hard in the general case [Karp, 1972]. In this problem we are given a $k \times k \times k$ cost matrix $C = (c_{ijl})$, and we have to select k elements of C, so that there is exactly one element in each two-dimensional face (in the three orientations), and the sum of the corresponding costs is minimized.

In this chapter we address the TA problem in the Co-MAPF context and formulate it as an axial 3-index assignment problem. We define the *global cooperation cost matrix* containing the total cost of all possible combinations of agents and tasks, and without considering possible conflicts between agents. Given the global cooperation cost matrix, we propose to use an off-the-shelf approximate solver to solve the task assignment problem.

We then wish to solve a more complete version of the Co-MAPF problem, where we also need to assign agents to tasks. More specifically, we need to (i) pair initiator agents with executor

agents and assign a task to each pair, (ii) determine meetings for each pair, and (iii) find conflict-free paths, such that the total cost is minimized. We suggest two possible approaches to solve the task assignment as a part of Co-CBS, i.e., (i) a *single-shot* approach where we find a single (approximately) optimal assignment, and then solve the induced Co-MAPF problem as before, and (ii) a *full-search* approach where we search over the space of assignments during the search for meetings and paths.

To measure the performance of our suggest Co-CBS with task assignment and evaluate the efficiency of the off-the-shelf assignment algorithm, we also suggest a fast greedy-assignment algorithm (described in Section 5.4) to be used with Co-CBS in the single-shot approach. In the next chapter, we present an empirical evaluation which, among others, compares the different approaches to solving the TA problem.

5.1 Assignment Problems

5.1.1 Two-Dimensional Assignment Problems

The task assignment problem in the classical MAPF problem is a two-dimensional assignment problem. This problem is known as the *Linear Assignment Problem* [Burkard et al., 2009], which can formulated as a constrained optimization problem. More specifically, by introducing binary matrix $X = (x_{ij})$ such that

$$x_{ij} = \begin{cases} 1, & \text{if row } i \text{ is assigned to column } j \\ 0, & \text{otherwise} \end{cases}, \tag{5.1}$$

the linear assignment problem can be modeled as:

$$\min_{x_{ij}} \sum_{i=1}^{k} \sum_{j=1}^{k} c_{ij} x_{ij}, \tag{5.2}$$

subject to

$$\begin{aligned} \sum_{j=1}^{k} x_{ij} &= 1; \qquad i = 1, 2, \ldots, k, \\ \sum_{i=1}^{k} x_{ij} &= 1; \qquad j = 1, 2, \ldots, k, \\ x_{ij} &\in \{0, 1\}; \qquad i, j = 1, 2, \ldots, k. \end{aligned}$$

The linear assignment problem can be solved optimally in polynomial time using *The Hungarian Algorithm* [Kuhn, 1955]. CBS-TA [Hönig et al., 2018] uses an extension to the Hungarian algorithm, called *Murty*'s algorithm [Murty, 1968] that can find the $m-$best assignments, and generate assignments with non-decreasing costs.

5.1.2 The Multi-Index Assignment Problem and the Axial 3-Index Assignment Problem

Task assignment for Co-MAPF can be formulated as a special case of the axial three-dimensional assignment problem [Spieksma, 2000], where the number of elements is the same on all dimensions. We are given a $k \times k \times k$ cost matrix $C = (c_{ijl})$, and we want to select k elements of C, so that there is exactly one element in each two-dimensional face (in the three orientations), and the sum of the corresponding costs is minimized. This problem can be formulated as follows:

$$\min_{x_{ijl}} \sum_{i=1}^{k} \sum_{j=1}^{k} \sum_{l=1}^{k} c_{ijl} x_{ijl}, \tag{5.3}$$

subject to

$$\begin{aligned}
\sum_{j=1}^{k} \sum_{l=1}^{k} x_{ijl} &= 1; \qquad i = 1, 2, \ldots, k, \\
\sum_{i=1}^{k} \sum_{l=1}^{k} x_{ijl} &= 1; \qquad j = 1, 2, \ldots, k, \\
\sum_{i=1}^{k} \sum_{j=1}^{k} x_{ijl} &= 1; \qquad l = 1, 2, \ldots, k, \\
x_{ijl} &\in \{0, 1\}; \qquad i, j, l = 1, 2, \ldots, k.
\end{aligned}$$

The axial three-dimensional assignment problem is a special case of the more general multi-index assignment problem (MIAP), also known as the S-dimensional (S-D) assignment problem, which can be formulated as follows:

$$\min_{x_{i_1 i_2 \ldots i_S}} \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_S=1}^{n_S} c_{i_1 i_2 \ldots i_S} x_{i_1 i_2 \ldots i_S}, \tag{5.4}$$

subject to

$$\begin{aligned}
\sum_{i_2=1}^{n_2} \cdots \sum_{i_S=1}^{n_S} x_{i_1 i_2 \ldots i_S} &= 1; \qquad i_1 = 1, 2, \ldots, n_1, \\
\sum_{i_1=1}^{n_1} \cdots \sum_{i_S=1}^{n_S} x_{i_1 i_2 \ldots i_S} &= 1; \qquad i_2 = 1, 2, \ldots, n_2, \\
&\vdots \\
\sum_{i_1=1}^{n_1} \cdots \sum_{i_{S-1}=1}^{n_{S-1}} x_{i_1 i_2 \ldots i_S} &= 1; \qquad i_S = 1, 2, \ldots, n_S, \\
x_{i_1 i_2 \ldots i_S} &\in \{0, 1\}; \qquad i_1, i_2, \ldots i_S = 1, 2, \ldots, n_S
\end{aligned}$$

We also present the MIAP formulation since the algorithm used to solve the three-dimensional problem (presented in the next section) also solves the S-D problem. This means that our suggested approach for solving the task assignment problem can be extended to solve a more

general version of Co-MAPF, where more than two agents work together on a single task. This generalization is discussed in Chapter 7.

5.1.3 Approximating the m-Best Solutions

To solve the 3-D assignment problem, we use an off-the-shelf algorithm [Deb et al., 1997; Popp et al., 2001], originally used in the field of *multi-target tracking*. Generally speaking, it uses a *Lagrange Relaxation* technique to approximate the optimal assignment [Deb et al., 1997], and then a variation of Murty's algorithm [Murty, 1968], to approximate the m-best assignments. The number of desired assignments m may affect the approximation accuracy, as a larger m value can yield lower-cost (i.e., better) assignments. For more details please refer to [Popp et al., 2001]. We'll refer to the algorithm as `compute_assignments(C,m)` where C is the $k \times k \times k$ cost matrix and m is the number of wanted assignments.

5.2 Cooperative Task Assignment as a 3-D Assignment Problem

5.2.1 The Global Cooperation Cost Matrix

We start by defining several notations. An assignment for task τ_i, is a tuple

$$\langle \tau_i, \alpha_j, \beta_l \rangle , \tag{5.5}$$

such that initiator agent α_j and executor agent β_l are working together on task τ_i. We denote an assignment in short by $\phi_{ijl} = \langle \tau_i, \alpha_j, \beta_l \rangle$. The cost of assignment ϕ_{ijl}, denoted c_{ijl}, is defined as the cost of completing task τ_i by agents α_j and β_l via the lowest-cost meeting (i.e., the lowest-cost vertex when using the earliest possible meeting time for each vertex). We denote this meeting by m^*_{ijl} and calculate its costs using

$$c_{ijl} = \min_v C_i \left(v, t^*_{ijl}(v) \right) \underbrace{=}_{Eq.\ 4.1} \begin{cases} 2 \cdot t + d(v, g_i), & t \geq t^*_{ijl}(v) \\ \infty, & \text{otherwise} \end{cases} , \tag{5.6}$$

where

$$t^*_{ijl}(v) = \max \left\{ d\left(\mathcal{V}_0(\alpha_j), s_i\right) + d(s_i, v), d\left(\mathcal{V}_0(\beta_l), v\right) \right\} . \tag{5.7}$$

Note that this cost is calculated without considering possible conflicts between agents. Therefore, the cost of a given assignment (of a single task) is a lower bound of the real cost using conflict-free paths.

Φ is a set of assignments, one for each task, namely $\Phi = \{\phi_{1j_1l_1}, \ldots, \phi_{kj_kl_k}\}$, such that $\phi_{ij_il_i}$ means assigning task τ_i to agents α_{j_i} and β_{l_i}. The cost of a set of assignments is the sum of costs of all assignments, namely

$$cost(\Phi) = \sum_{i=1}^{k} c_{ij_il_i}. \tag{5.8}$$

The cost of the set of assignments is a lower bound of the real solution cost (i.e., the sum of costs of all agents), in which we consider conflict-free paths.

The global cooperation cost matrix is a matrix

$$C : \mathcal{T} \times \mathcal{A} \times \mathcal{B} \to \mathbb{R} \tag{5.9}$$

such that $C(\tau_i, \alpha_j, \beta_l) = c_{ijl}$. Matrix C has k^3 entries, and for each entry, we need to compute a Meetings Table (i.e., find the optimal meeting) for each task by running Dijkstra three times. Thus, computing C is of time complexity

$$\mathcal{O}\left(k^3 \cdot |\mathcal{T}| \cdot (|V|log|V| + |E|)\right) = \mathcal{O}\left(k^4 \cdot (|V|log|V| + |E|)\right). \tag{5.10}$$

However, in the global cooperation matrix we need to account only for the best meeting (for each possible assignment), i.e., a meeting at a specific location at the earliest time possible. This means that the calculation can be decoupled between agents. Namely, we can run Dijkstra for each task independently to find paths to all the vertices in the graph, and then iterate on all possible assignments and for each one on all possible vertices by using Equations 5.6 and 5.7. Overall, we get a time complexity of

$$\mathcal{O}\left(k \cdot (|V|log|V| + |E|)\right) + \mathcal{O}\left(k^3 \cdot |V|\right) \tag{5.11}$$

for computing the global cooperation cost matrix. The first term is running 3 times Dijkstra's algorithm for each task (i.e., from the task start and goal locations, and from the executor agent start location). The second term is for checking all possible meeting locations for each possible assignment.

Note that while there are k^3 different combinations of agents and tasks, for the 3-D problem there are $(k!)^2$ different sets of assignments, i.e., possible solutions to Equation 5.3. It is therefore not feasible to find the optimal set of assignments in a brute-force manner for large values of k.

5.3 Cooperative Conflict-Based Search with Task Assignment

In this section we describe two possible approaches to incorporate the assignment algorithm (described in Section 5.1.3) into Co-CBS for solving the Co-MAPF problem with task assignment. Both approaches use the global cooperation cost matrix C (described in Section 5.2.1) and `compute_assignments(C,m)` (described in Section 5.1.3) to calculate assignments. More specifically, the output of `compute_assignments(C,m)` is a vector of sets of assignments $\{\Phi_0, \ldots, \Phi_m\}$, ordered by cost, namely

$$cost(\Phi_0) \leq cost(\Phi_1) \leq \cdots \leq cost(\Phi_m). \tag{5.12}$$

5.3.1 Single-Shot Task Assignment (Single-TA)

The first approach for solving Co-MAPF with TA is straightforward: approximate the optimal assignment given C and solve the induced Co-MAPF problem with Co-CBS. Note that when computing the (approximate) optimal assignments, we may use different values of m, to get the m-best assignments and then select the lowest-cost assignment. Namely, we compute $\{\Phi_0, \ldots, \Phi_m\}$ and use only Φ_0.

5.3.2 Full Task-Assignment Search (Full-TA)

Rather than using a single assignment, we may also search the *assignments space* (i.e., the space of all possible assignments) in a best-first manner, similarly to CBS-TA [Hönig et al., 2018]. Since Co-CBS already searches over a forest of CTs, Co-CBS with Full-TA searches over a forest of forests. Namely, for each assignment it constructs a forest of CTs and performs a Co-CBS search. For a given set of assignments $\Phi = \{\phi_{1j_1l_1}, \ldots, \phi_{kj_kl_k}\}$ we denote $\mathcal{M}^*(\Phi)$ as the *base* set of meetings, namely the lowest cost meeting of each assignment:

$$\mathcal{M}^*(\Phi) = \left\{m^*_{1j_1l_1}, \ldots, m^*_{kj_kl_k}\right\}. \tag{5.13}$$

We then define a *base root node* as a Co-CBS root node with the lowest-cost set of meetings $\mathcal{M}^*(\Phi)$ for a specific assignment. The size of the assignments space is determined by m, which is given as an input to the algorithm. Given m, Co-CBS with Full-TA calculates the (approximate) m-best assignments $\{\Phi_0, \ldots, \Phi_m\}$, which are stored in a non-decreasing order. Each CT node has an additional field *assignment*, which specifies a set of assignments, one for each task. The first generated root node R_0 is a base root node with the lowest-cost assignment found, and the corresponding lowest-cost set of meetings, namely $R_0.assignment = \Phi_0$ and $R_0.meetings = \mathcal{M}^*(\Phi_0)$. Every time a base root node is selected for expansion, in addition to creating regular nodes (due to a conflict) and root nodes (with the next best sets of meetings), we also create a new base root node with the next best assignment. This is described in Algorithm 5.1, with the changes from the `expand_root` procedure (described in Algorithm 4.3) highlighted. Since the size of the assignments space is pre-determined by m, Co-CBS with Full-TA only creates up to m base root nodes (corresponding to m sets of assignments). Therefore, when selecting for expansion the base root node with the last set of assignments Φ_m, no new base root nodes are created (lines 3-4 in Algorithm 5.1). Otherwise, we create a new base root node with the set of assignments and its corresponding lowest-cost set of meetings (lines 7-8), and plan new paths for all agents, since the assignment has changed (line 10). When creating new root nodes with different sets of meetings, we use the same assignment as the parent node (line 16).

Note that the parameter m defines the size of the assignments space. Namely, it may affect both the solution quality and run time. In our empirical evaluation, presented in the next chapter, we show results for different m values.

5.4 Greedy Assignment Approach

Computing assignments using the `compute_assignments(C,m)` algorithm presented in Section 5.1.3 requires the computation of the global cooperation cost matrix, which can be computationally expensive for large values of k (see Section 5.2.1). Therefore, to evaluate the improvement achieved using the assignment algorithm, we also propose a greedy-assignment approach, that would be sub-optimal in terms of cost, but much faster to compute.

The greedy approach exploits the fact that in the definition of the meeting (and assignment) cost (Equations 5.6 and 5.7), the two agents are loosely coupled only via the meeting time. It

Algorithm 5.1 Expand base root

1: **Input:** $\{\Phi_0, \ldots, \Phi_m\}$, Meetings tables, a root node P with $P.assignment = \Phi_j$
2: **if** P is base root **then** ▷ namely, $P.meetings = \mathcal{M}^*(\Phi_j)$
3: **if** $j = m$ **then** ▷ no more sets of assignments exist
4: break
5: Q = new node
6: $Q.constraints = \emptyset$
7: $Q.assignment = \Phi_{j+1}$
8: $Q.meetings = \mathcal{M}^*(\Phi_{j+1})$ ▷ lowest-cost set of meetings for new set of assignments
9: $Q.root$ = True
10: $Q.solution = plan\_paths()$ ▷ plan all paths since assignments changed
11: $Q.cost = compute\_cost(Q.solution)$
12: insert Q to OpenRoots
13: **for all** $\tau_i \in \mathcal{T}$ **do** ▷ loop over all tasks
14: R = new node
15: $R.constraints = \emptyset$
16: $R.assignment = P.assignment$ ▷ new root nodes has the same assignments as P
17: $R.meetings = P.meetings$
18: $R.meetings[\tau_i] = get\_next\_meeting(T_i)$ ▷ change only τ_i's meeting
19: $R.root$ = True
20: Update $R.solution$ by invoking $plan\_paths(\alpha_i, \beta_i)$
21: $R.cost = compute\_cost(R.solution)$
22: insert R to OpenRoots

is therefore reasonable to select an agent of each type for each task independently. Thus, the greedy approach loops over all tasks, and for each task computes the shortest path for each agent (without considering the other agents), and selects the closest initiator agent, and then the closest executor agent, out of all agents that haven't been selected yet. This approach is depicted in Algorithm 5.2.

In our empirical evaluation, presented in the next chapter, we use the greedy-assignment approach with Co-CBS in the single-shot approach (see Section 5.3.1).

To summarize, in this chapter we addressed the task assignment (TA) problem in the context of the Co-MAPF framework. We showed how to formulate it as a multi-index assignment problem and proposed to use and off-the-shelf approximate algorithm by incorporating it into Co-CBS in two different ways. We've also suggested a greedy assignment algorithm, which will be used as a baseline in our empirical evaluation, presented in the next chapter. We haven't discussed the theoretical properties and implications of solving the TA problem, in terms of solution optimality and run time. We leave this discussion for future work.

Algorithm 5.2 Greedy Cooperative Task Assignment

1: $assigned_\alpha = \{1, \ldots, k\}$ ▷ initialize array for assigned initiator agents
2: $assigned_\beta = \{1, \ldots, k\}$ ▷ initialize array for assigned executor agents
3: **for all** τ_i in $\mathcal{T}$ **do**
4: Reset d_α and d_β
5: **for all** α_j in $\mathcal{A}$ **do**
6: $d_\alpha[j] = d^*\left(s_i, \mathcal{V}_0\left(\alpha_j\right)\right)$ ▷ shortest path from agent α_j start to task τ_i start location
7: **for all** β_j in $\mathcal{B}$ **do**
8: $d_\beta[j] = d^*\left(s_i, \mathcal{V}_0\left(\beta_j\right)\right)$ ▷ shortest path from agent β_j start to task τ_i start location
9: $closest_\alpha = \underset{j \in assigned_\alpha}{\arg\min}\ d_\alpha$
10: $closest_\beta = \underset{j \in assigned_\beta}{\arg\min}\ d_\beta$
11: Remove $closest_\alpha$ from $assigned_\alpha$
12: Remove $closest_\beta$ from $assigned_\beta$
13: Assign τ_i to $(closest_\alpha, closest_\beta)$

Chapter 6

Experimental Evaluation

In previous chapters, we introduced the Co-MAPF framework and suggested the Co-CBS algorithm for solving Co-MAPF problem instances. To the best of our knowledge, there does not exist an off-the-shelf optimal solver for MAPF problems involving cooperative behavior in the form introduced in this work. Therefore, it is not possible to measure the performance of Co-CBS against an off-the-shelf algorithm. Moreover, in Chapter 4, we've discussed some of the difficulties that rise when attempting to solve the Co-MAPF problem using a centralized A*-based implementation. An attempt to solve Co-MAPF using such implementation would yield similar results as solving classical MAPF problem using A* [Sharon et al., 2015], due to their similar search approach and conflict-resolution mechanism.

We do however wish to measure the quality of Co-CBS in this chapter. We achieve that by performing two different experimental evaluations. First, we present the results of running Co-CBS on standard MAPF benchmarks [Stern et al., 2019; Sturtevant, 2012]. We compare the performance of the basic version of Co-CBS as well as with the suggested improvements (described in Section 4.5). We demonstrate the scalability of Co-CBS and discuss the effect of adding cooperative behavior on the performance. Second, we compare Co-CBS with a baseline prioritized planning algorithm. The baseline algorithm plans for each agent independently, considering the paths of previous agents, and using the optimal meeting location. Thus, it runs very fast, albeit it is sub-optimal.

Finally, we perform an evaluation of solving the Co-MAPF problem with task assignment, comparing the different approaches discussed in Chapter 5.

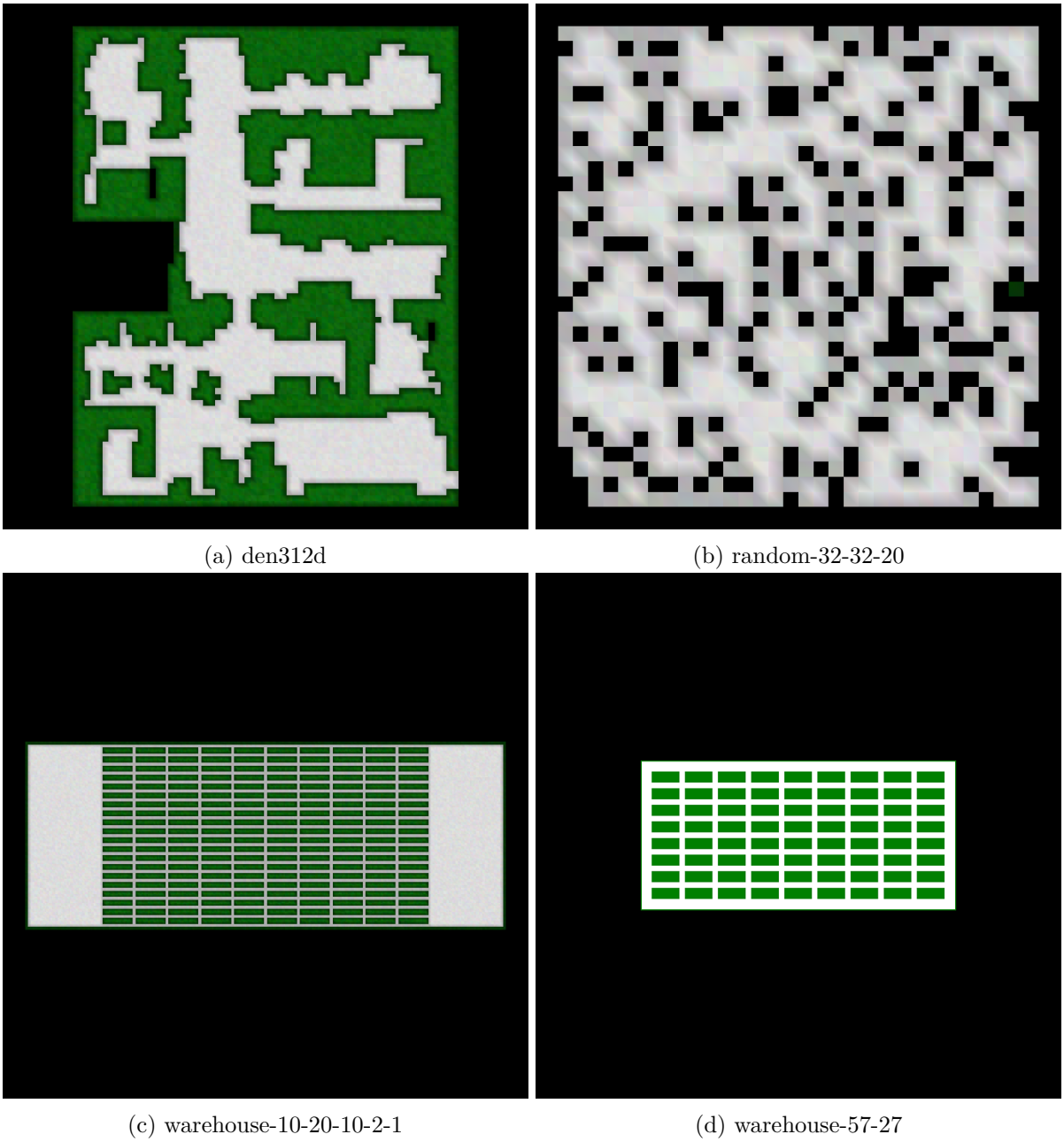
The empirical evaluation shows that Co-CBS solves non-trivial Co-MAPF instances on standard MAPF benchmarks, and that our two suggested improvements significantly improve the algorithm's performance. We also show that also solving the task assignment problem may dramatically improve the solution cost, as well the algorithm's performance.

6.1 Setup and Benchmarks

Co-CBS is implemented in C++ based on the implementation of Li et al. [Li et al., 2021]. The source code is available at https://github.com/CRL-Technion/Cooperative-MAPF. All simulations were performed on an Intel Xeon Platinum 8000 @ 3.1Ghz machine with 32.0 GB RAM.

We used four benchmark environments for our empirical evaluation. Three standard MAPF

benchmarks [Stern et al., 2019; Sturtevant, 2012], and a custom small warehouse environment we've created for the evaluation. The benchmarks are depicted in Figure 6.1 and described in Table 6.1. For each benchmark we specify the size of the grid, as well as the number of free (non-obstacle) cells (i.e., the number of vertices in the graph). We also state the ratio between the number of free cells and total cells, which is a (crude) measurement indicating how "sparse" the environment is[1].



(a) den312d (b) random-32-32-20

(c) warehouse-10-20-10-2-1 (d) warehouse-57-27

Figure 6.1: Benchmark maps.

We ran 25 random queries for each benchmark for the SOC objective with the number of tasks ranging from 6 tasks (12 agents) to 22 tasks (44 agents) and with a time limit of two

[1]The precise notion of a sparse environment requires introducing additional definitions which are out of the context of this thesis but here we use the term intuitively.

Benchmark name	den312d	random-32-32-20	warehouse-10-20-10-2-1	warehouse-57-27
Environment type	video game	random	warehouse	warehouse
Grid Size	$65 \times 81 = 5,265$	$32 \times 32 = 1,024$	$161 \times 63 = 10,143$	$57 \times 27 = 1,539$
# vertices	2,445	819	5,699	819
Sparsity ratio	0.46	0.8	0.56	0.53

Table 6.1: Benchmarks description.

minutes. For each benchmark, we compare the performance of three different variants of Co-CBS: (i) basic Co-CBS, (ii) Co-CBS with prioritizing conflicts (PC), and (iii) Co-CBS with PC and lazy expansion (LE) of root nodes.

As opposed to classical MAPF, where each agent is provided with start and goal locations, in Co-MAPF, a task's start and goal need to be provided (instead of explicitly providing an agent's goal). Thus, we defined the tasks in each scenario as follows, based on the original benchmark scenario: for each pair of agents, one set of start and goal locations is used for the task, and the other set is used for the agents' start locations.

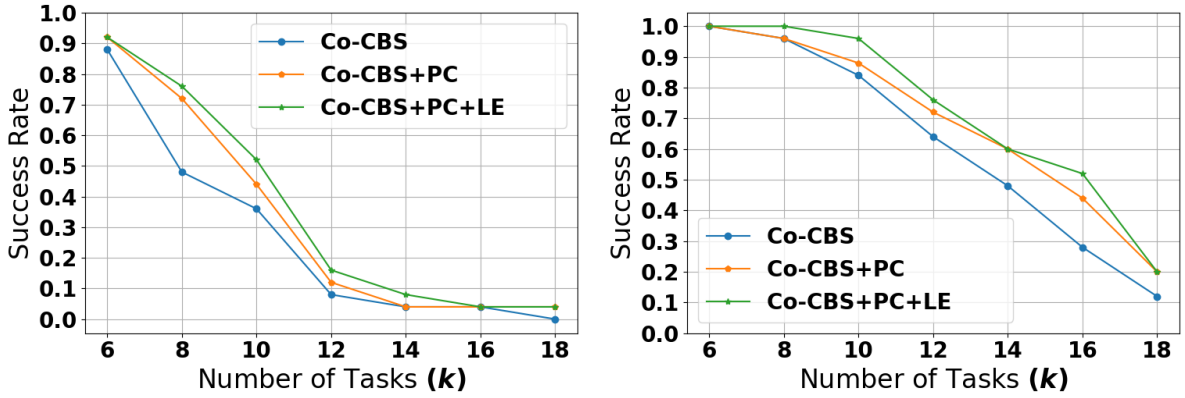
6.2 Co-CBS Empirical Evaluation

We first examine the algorithm's success rate (i.e., the ratio of solved instances within the time limit) for all benchmarks. Figure 6.2 shows the success rates of Co-CBS on all benchmarks. Co-CBS successfully solves more than 80% of the instances with ten tasks on all benchmarks besides `den312d`, in which the success rate sharply drops below 20% for twelve tasks or more. This map is very dense, where most of the cells are blocked. When adding more agents it becomes even denser, which explains the drop in the success rate. The random map is very sparse, but very small. Therefore, the success rate drops for 18 tasks. On the other hand, on the large warehouse (`warehouse-10-20-10-2-1`) Co-CBS achieves more than 50% success rate on 20 tasks (40 agents).

Using PC improves the basic version of Co-CBS in all cases, achieving up to 30% increase in the success rate. Furthermore, adding LE on top of PC further improves the performance in most cases, and never degrades the performance. This is especially notable with a large number of tasks, where many root nodes are created.

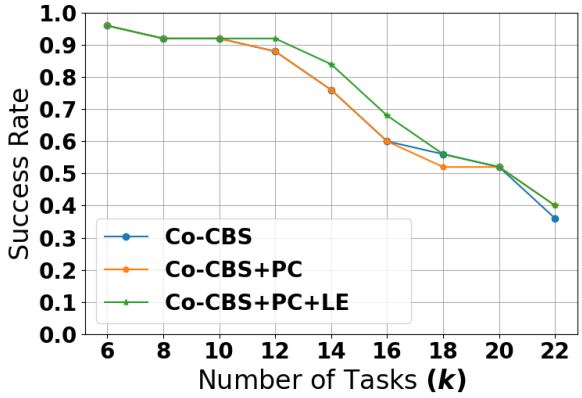
We now wish to examine the search in the meetings space. Figure 6.3a shows the average number of generated meeting sets, which is equivalent to the number of generated root nodes. Note that higher values mean more cardinal conflicts that cause an increase in the solution cost. Therefore Co-CBS chooses to expand the search in the meetings space, by selecting root nodes for expansion, and generating more root nodes (and meeting sets). This is especially notable on the random and small warehouse benchmarks and correlates with the results shown in Figures 6.2b and 6.2d, where adding PC+LE significantly improves the success rate.

Figure 6.3b shows the ratio η between the number of instances where the first set of meetings is used to obtain the solution and the total number of instances. Both warehouse environments are typically sparser, causing fewer conflicts between agents. Thus, for small number of tasks, a feasible solution is usually found quickly using the first set of meetings. This is especially notable in the large warehouse, where most solutions are obtained using the first set of meetings (i.e.,

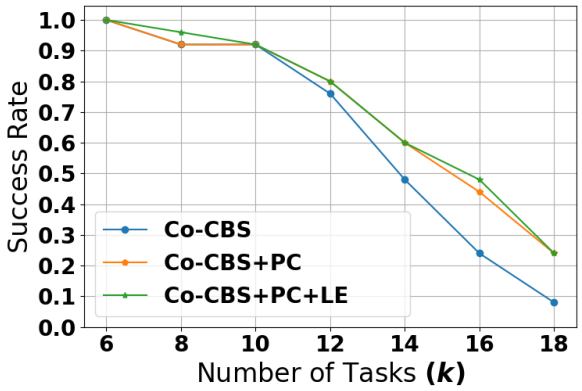


(a) den312d

(b) random-32-32-20



(c) warehouse-10-20-10-2-1



(d) warehouse-57-27

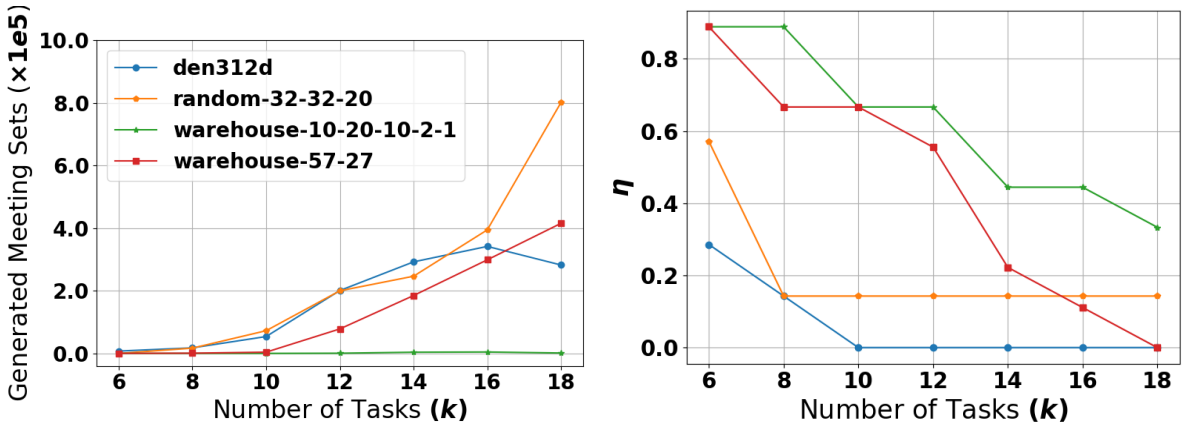
Figure 6.2: Success rates.

when η is close to one). The search in this case is equivalent to running CBS with the first set of meetings. For the same reason, adding PC+LE does not significantly improve the performance in this environment (as seen in Figure 6.2c). On the other hand, in other smaller and denser environments, most solutions are not obtained using the first generated set of meetings (i.e., η approaches 0, especially for a large number of tasks). A more exhaustive meeting-space search is therefore required to find an optimal solution, as shown in Figure 6.3a. In these cases, the success rate drops accordingly.

Figure 6.4 shows the results achieved using the baseline planner. We state that the baseline algorithm achieves 100% success rate on all benchmarks. However, it performs very poorly in terms of solution costs. The large warehouse environment is large and sparse, therefore the baseline planner manages to plan conflict-free paths while suffering only a minor increase to the cost. However, on the other maps the baseline planner suffers a 10% cost increase already for only six tasks, the cost is increased by up to 40%.

6.3 Co-CBS with Task Assignment

In this section we present the results of an empirical evaluation for solving the Co-MAPF with task assignment problem presented in Chapter 5. In this problem we do not assume that tasks



(a) (b)

Figure 6.3: (a) Number of generated sets of meetings. (b) Ratio η between the number of instances solved using the first set of meetings, and the total number of instances.

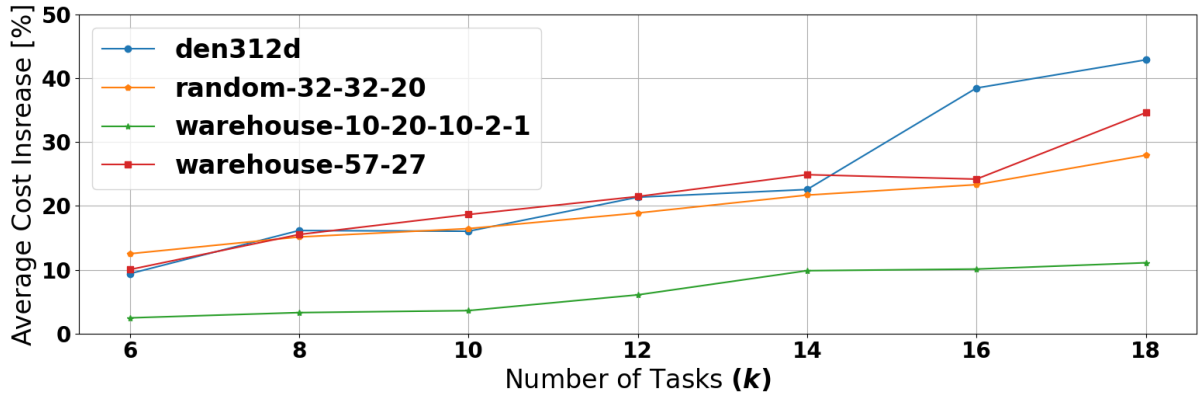


Figure 6.4: The average cost increase over the optimal cost using the (sub-optimal) baseline planner.

are pre-assigned to agents, and we also wish to solve the corresponding task assignment problem. More specifically, we wish to determine the pairing of initiator agents with executor agents, and pairs of agents to tasks, such that the total cost is minimized.

In the previous section we presented empirical results of running different variants of Co-CBS without solving the task assignment problem. In practice, we used an arbitrary random assignment of agents and tasks. In this section, we wish to empirically evaluate the effect of solving the assignment problem for Co-MAPF. We compare the cost of obtained solutions without solving the task assignment problem, with the different approaches for solving the task assignment problem as well (see Section 5.3).

For the evaluation we use the same setup and benchmarks described in Section 6.1, and run the improved version of Co-CBS, namely using PC+LE (see Section 4.5).

Recall that the purpose of solving the task assignment problem is to minimize the total cost (see Equation 5.3). However, we first present and discuss the effect of solving the task assignment problem on the algorithm's success rate. In this section we use a five minutes timeout for all experiments, and present the results of five different algorithms:

1. **Co-CBS**. Namely Co-CBS with an arbitrary assignment as in Section 6.2.

2. **Co-CBS with Greedy Assignment**. Namely, Co-CBS with a single-shot greedy assignment (see Section 5.4).

3. **Co-CBS with Single Assignment ($m = 1$)**. Namely Co-CBS with a single-shot assignment using the `compute_assignments` algorithm with $m = 1$ (see Sections 5.1.3 and 5.3.1).

4. **Co-CBS with Single Assignment ($m = 100$)**. Namely, Co-CBS with a single-shot assignment using the `compute_assignments` algorithm with $m = 100$.

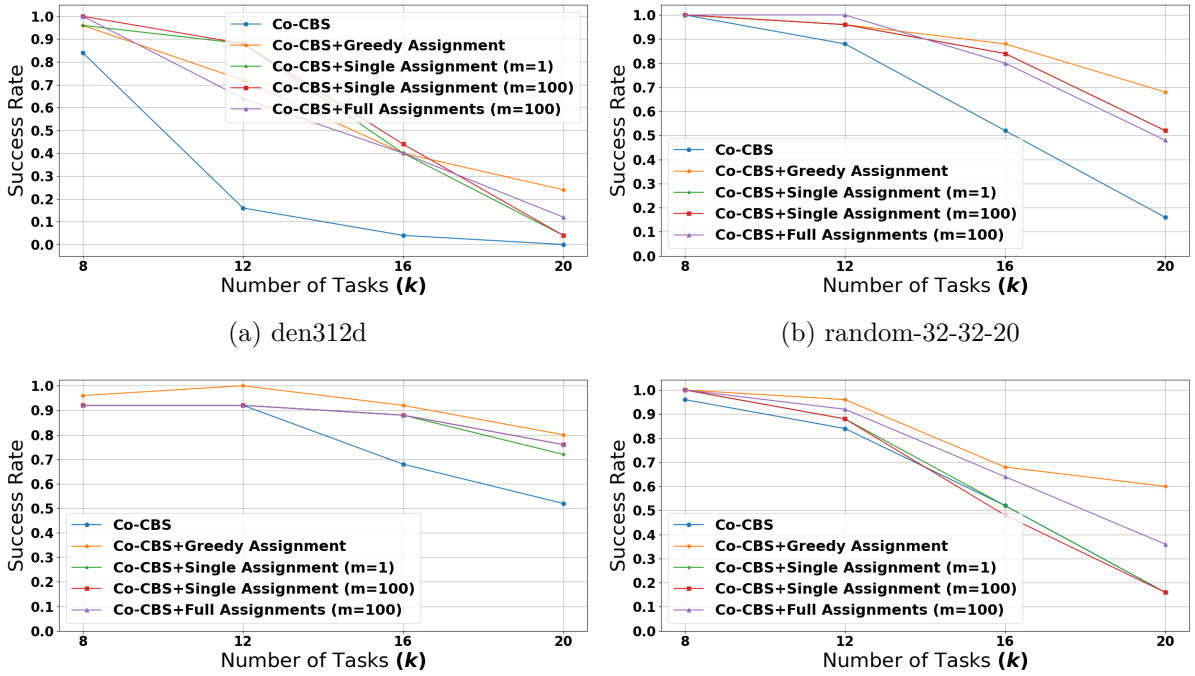
5. **Co-CBS with Full Assignment Search**. Namely, Co-CBS with a full assignments search, with $m = 100$ (see Section 5.3.2).

Figure 6.5 shows the success rates of all algorithms on all benchmarks. We see that using any approach for assigning agents and tasks (as opposed to a random assignment shown by the blue line) may cause a significant increase in the success rate, up to roughly 70% in some of the cases. The highest success rate is usually achieved using the greedy assignment algorithm, as it does not require the calculation of the global cooperation cost matrix (see Section 5.2.1), while still solving the assignment problem (albeit not optimally). On the small warehouse map (`warehouse-57-27`) and the `den312d` map, which are very dense for a large number of agents, the full-assignments algorithm achieves the best success rates (after the greedy assignment). This is due to the fact that we consider solutions with different sets of assignments (rather than just one), thus we find a solution more quickly.

There can be several reasons why solving the task assignment problem improves the success rate compared to an arbitrary assignment. The main reason is that a good assignment makes agents work in local environments thus avoiding many unnecessary conflicts. A further investigation of this phenomena is discussed as an interesting direction for future research in Chapter 7.

We now wish to examine the cost of the obtained solutions using the different algorithms. Figure 6.6 shows several representative cases where we examine both the success rate and the cost of the obtained solutions. The results are presented on a bi-objective graph, where the x axis is the average cost increase rate (compared to the lowest-cost solution found for each scenario). The y axis is the average *fail rate* (i.e., $(1 - success\_rate)$). Better solutions have lower average cost increase rate, as well as low fail rate. The blue circle, representing Co-CBS without task assignment, is higher and to the right than all other algorithms in all cases. This means that it performs the worst both in terms of cost and success rate. The greedy-assignment algorithm achieves better success rates in most cases, but it returns solutions with higher cost. We also see that there are only minor differences between the different algorithms that use the `compute_assignments` algorithm, and that all of them outperforms the greedy-assignment algorithm in terms of the cost. Note that in the case of the large warehouse and 40 agents (Figure 6.6c), the cost of the obtained solutions using our suggested assignment algorithms is almost half of the cost using an arbitrary assignment.

Finally, we present the time it takes the different algorithms to solve problem instances for different number of tasks, in a representative case, and broken down into the different parts of the algorithms. Figure 6.7 shows the total average time it takes each algorithm to solve problem instances of the large warehouse map (`warehouse-10-20-10-2-1`). The total time refers only



(a) den312d

(b) random-32-32-20

(c) warehouse-10-20-10-2-1

(d) warehouse-57-27

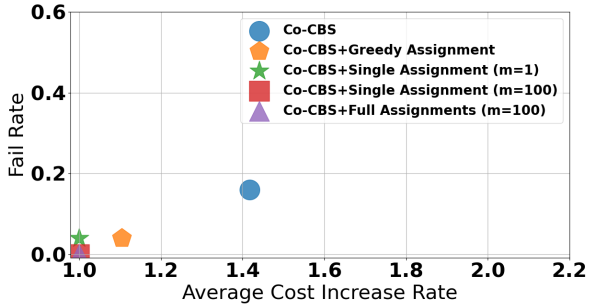
Figure 6.5: Success rates solving the task assignment problem.

to solved instances (which were solved within the time limit), and it is divided to three different parts:

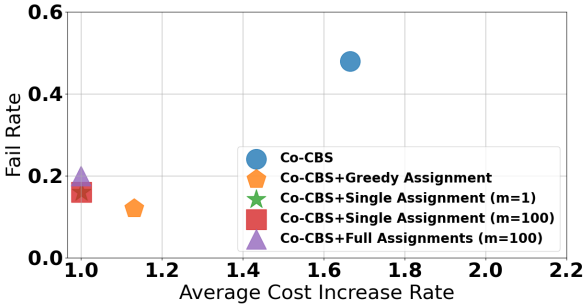
1. In blue, the time it takes to compute the global cooperation cost matrix (see Section 5.2.1).
2. In orange, the time it takes to compute the assignments using the `compute_assignments` algorithm (see Section 5.1.3).
3. In green, the time it takes to run Co-CBS, given the assignment(s).

Co-CBS without task assignment (Figure 6.7a) and Co-CBS with a greedy assignment (6.7b) solve only trivial instances that can be solved within a few seconds. Calculating the greedy assignment is very fast it and takes very little time. When using the `compute_assignments` algorithm for a large k, we see that most of the time is spent on calculating the global cooperation cost matrix, and a significant amount of time on the `compute_assignments` algorithm (when $m = 100$). This is the main reason why the greedy-assignment algorithm achieves a higher success rate. For large values of k, we are highly motivated to reduce the time it takes to compute the cost matrix, and indeed we suggest such an approach as a future research direction in Chapter 7.

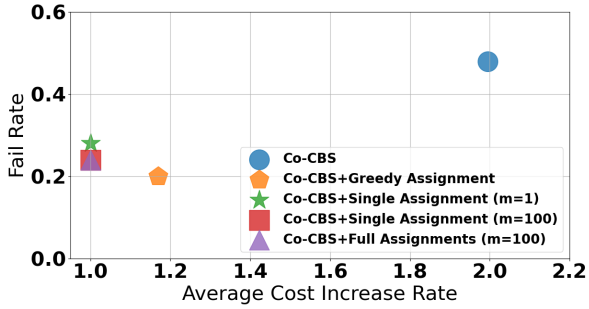
To summarize, in this chapter we presented a comprehensive empirical evaluation of our proposed Co-CBS algorithm for solving the Co-MAPF problem. We presented the results of running three different variants of Co-CBS, and saw that it outperforms a baseline prioritized planner in terms of cost. In addition, we provided an empirical analysis demonstrating the search Co-CBS performs in the meetings space for different cases. Finally, we presented results of solving the task assignment problem, and we conclude that solving the task assignment as well may significantly improve both the solution cost as well as the algorithm's performance.



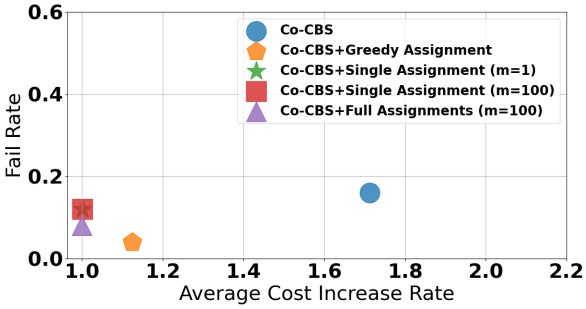
(a) den312d, $k = 8$



(b) random-32-32-20, $k = 16$

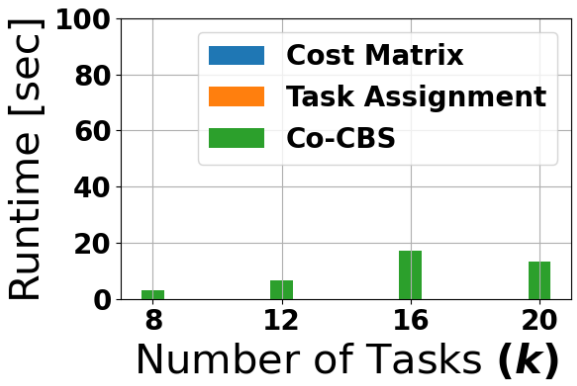


(c) warehouse-10-20-10-2-1, $k = 20$

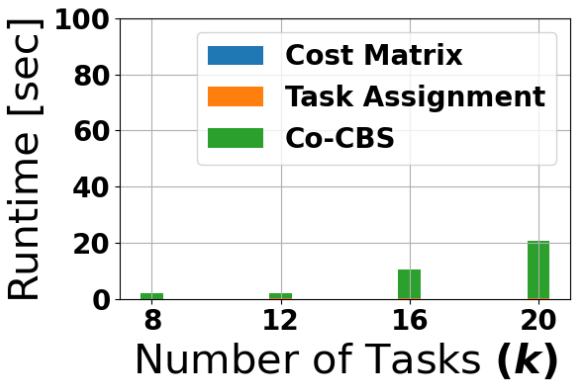


(d) warehouse-57-27, $k = 12$

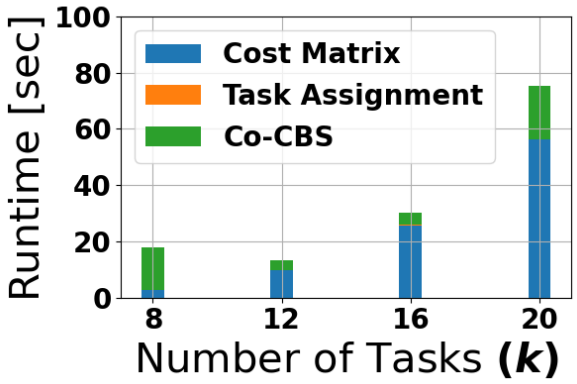
Figure 6.6: Success rates solving the task assignment problem.



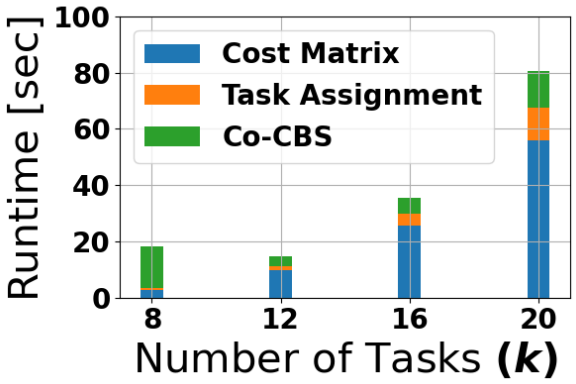
(a) Co-CBS



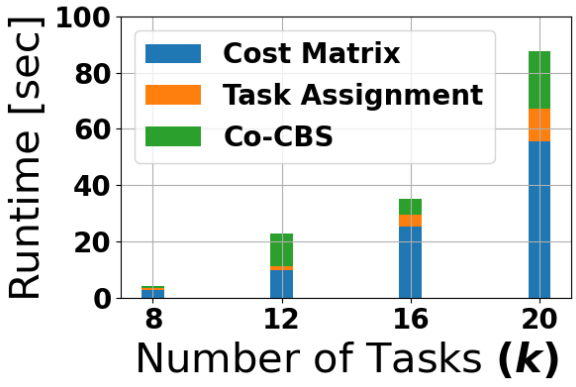
(b) Co-CBS + Greedy Assignment



(c) Co-CBS + Single Assignment ($m = 1$)



(d) Co-CBS + Single Assignment ($m = 100$)



(e) Co-CBS + Full Assignments ($m = 100$)

Figure 6.7: Run-time of solving the task assignment problem.

Chapter 7

Conclusion and Future Work

In this final chapter we conclude our work by presenting a short summary and several directions for future work. Specifically, we provide a comprehensive discussion regarding the suggested model and algorithm, and suggest research directions in three different aspects: (i) improvements and extensions to the suggested Co-CBS algorithm, (ii) possible extensions to Co-MAPF framework, and (iii) open questions and issues regarding the task assignment problem in the Co-MAPF context.

7.1 Summary

In this paper, we introduced and studied the *Cooperative Multi-Agent Path Finding (Co-MAPF)* problem, an extension to classical MAPF that incorporates cooperative behavior. This problem is motivated by real-world scenarios where several *heterogeneous* agents must coordinate their decisions and actions to complete a joint task. We introduced a specific real-world problem, taken from the warehouse-automation domain, where two different types of robots work together to efficiently deliver packages in an automated warehouse.

We formally described the Co-MAPF problem, which is based on the classical MAPF formulation, and introduced the notion of *cooperative tasks*, which require the coordination of two different types of agents to be completed. Besides conflict-avoidance, we considered agents' interaction via *meetings*, and defined the way a cooperative task is completed by scheduling a meeting of the two agents.

Based on our suggested Co-MAPF formulation, we introduced *Cooperative Conflict-Based Search (Co-CBS)*, a three-level search algorithm that optimally solves Co-MAPF instances, by searching in a so-called *meetings space* while resolving conflicts between agents. More specifically, Co-CBS introduced the notion of *root nodes*, where each root node has a different set of meetings, one for each task. We've shown an efficient way to enumerate and generate sets of meetings in a best-first manner. Co-CBS searches the meetings space by expanding root nodes and generating new root nodes (with new sets of meetings), and performing a CBS-like search for each different set of meetings. We proved that Co-CBS finds the optimal solution for every solvable Co-MAPF problem instance. We also showed that it is complete on *source-connected* instances, a wide and general set of Co-MAPF problem instances.

Two improvements to Co-CBS were suggested: the previously suggested Prioritizing Conflict

(PC), and Lazy Expansion (LE) of root nodes. In PC, Co-CBS splits the search tree on *cardinal conflicts* first, which are found by constructing a *multi-valued decision diagrams (MDDs)*, which compactly store all valid paths of a given cost. We proposed an efficient method to construct MDDs in the Co-MAPF case, where agents are constrained to meet in order to complete a task. LE exploits special characteristic of root nodes for speeding up the search. More specifically, it uses the fact that the cost of sets of meetings is known in advance, therefore many low-level plan computations can be spared.

We addressed the *task assignment (TA)* problem for Co-MAPF. We formulated the problem as an *axial 3D assignment problem*, discussed its hardness and suggested to use an off-the-shelf algorithm to approximate the best assignments. We described several approaches to integrate task assignment into Co-CBS, and also suggested a greedy-assignment algorithm to use as a baseline.

In our empirical evaluation, we showed that Co-CBS can solve non-trivial instances on well-known MAPF benchmarks in a reasonable time. We showed how PC and LE improve the algorithm performance, and discussed the effect of searching for a meeting in different benchmarks. We also presented an empirical evaluation solving the task assignment problem, and showed it can dramatically improve the cost of obtained solutions, as well as the algorithm's success rate.

Co-MAPF is a newly-suggested framework that is based on the extensively-researched MAPF framework, towards more real-life applicability. We argue that Co-CBS forms a basic solution approach that may serve as a natural starting point for future extensions. In the next section we discuss several directions for future work.

7.2 Future Work

7.2.1 Co-CBS Improvements and Extensions

Node selection and expansion. Co-CBS may create a very large number of root nodes, as the meetings space is exponential in the size of the graph and number of tasks. It may be beneficial to use a heuristic function to guide the search towards better meeting sets, as well as a pruning mechanism to reject possibly-infeasible or sub-optimal meeting sets.

Information reusing between constraint trees. Co-CBS expands root nodes by only changing one meeting in the newly-created node. Moreover, the next selected meeting is usually very close to the current meeting, both in location and time. This implies that Co-CBS searches over multiple trees that potentially have very similar solutions. We may exploit this for more efficient computation.

Meetings-level search. Co-CBS uses a simple-yet-effective method for finding an optimal meeting for each task, by computing and storing a meetings table for each. For large problem instances, this may become memory and run-time expensive, due to the maintenance of large meeting tables. We may consider incorporating an algorithm such as the recently-proposed CF-MM* algorithm [Atzmon et al., 2021], for the Multi-Agent Meeting problem.

Two-level planning. Co-CBS decouples meetings planning from path planning by adding a third level of planning. Using CF-MM*, as suggested in the previous paragraph, to simultaneously search for a meeting for all agents can be used to handle conflicts between agents during the search for a meeting. Coupling the two may be advantageous as meetings and conflicts may be tightly coupled. Planning can then be done in two levels, where the low level handles both meetings and constraints.

Concurrent Planning and Parallelization. Co-CBS's search makes it a good candidate for parallelization. We may perform a parallel search on multiple conflict trees, which is equivalent to running multiple CBS searches simultaneously, each with a given set of meetings. Recall that when a root node is expanded, up to k new root nodes can be created, each one with only one meeting changed. This means that Co-CBS's OPEN list potentially contains a large number of nodes with a similar cost, at any given time. We can extract and validate all of them in parallel, and potentially save a lot of run time.

Existing CBS improvements. In addition to the PC improvement presented in Section 4.5, many more CBS improvements exist. Some of these include adding a heuristic function to the high-level search [Felner et al., 2018]; using positive constraints due to conflicts, also known as disjoint splitting [Li et al., 2019]; bypassing a conflict rather than splitting the search tree [Boyarski et al., 2015a]; symmetry breaking in agent paths (e.g., in corridors) [Li et al., 2020], and exploiting similarities between nodes in a single constraint tree [Boyarski et al., 2020]. We can also apply (bounded) sub-optimal variants of CBS [Barer et al., 2014] to Co-CBS.

Meetings in adjacent locations. Co-CBS solves the Co-MAPF problem where agents are required to meet in a single location. As discussed in Section 3.3.1, this definition of a meeting can be modified by requiring the agents to meet in adjacent locations (namely two vertices connected by an edge). Co-CBS can also solve these scenarios, with only slight modifications, as planning for each agent is done independently. More specifically, when computing a meetings table, for each vertex we need to account for all its neighbors and create a meeting for each one. Moreover, when planning paths, each agent would have to arrive at its corresponding meeting location (and time), rather than both agents arriving at a single location.

Co-CBS completeness. We defined the notion of source-connected Co-MAPF instances, a general and wide set of problem instances, and proved that Co-CBS is complete on these instances. However, Co-MAPF instances that do not adhere to the source-connected definition, are usually solvable, and specifically will be solved optimally by Co-CBS. We therefore need to find a more general way to check whether a Co-MAPF instance is solvable, similar to [Yu and Rus, 2014] for classical MAPF. This forms an interesting theoretical research question for future research.

7.2.2 Extensions to the Co-MAPF Framework

Number and types of collaborating agents. A rather straightforward generalization of the Co-MAPF framework is to require more than two agents to collaborate on a task. The problem

introduced in Chapter 1 motivates this extension: several grasp units may pickup several items for a single transfer unit. Co-CBS can solve this problem with a few minor changes. However, if the number of agents per task isn't fixed, additional work is required. Moreover, we may consider agents with different traversal capabilities (e.g., different velocities [Hönig et al., 2016]), by possibly changing the single-agent planner.

Other forms of cooperative interaction. We introduced a definition for the Co-MAPF problem, where interaction between agents is expressed via meetings between two types of agents, an initiator and an executor. While this interaction is very intuitive, more forms of cooperative interaction can be modeled. For example, we may represent temporal constraints between agents, such as precedence (e.g., the initiator need to arrive at the meeting location *before* the executor agent, rather than at the same time). We may generalize the formulation to include a finite set of possible agent types, and define more complex tasks where each agent type has its dedicated role. Furthermore, we state that the framework provided by Co-CBS might allow to address such general definitions by only adjusting the *cooperation-level* search (in our case, finding meeting locations and times, and constraining agents' paths to arrive at the meetings). Any cooperative planning, which results in inducing goals for an agent (for the path-level search), can be easily plugged in into Co-CBS.

Lifelong planning. In this problem we assume cooperative tasks are pre-defined and known in advance. However, *lifelong-planning* problems may fit to more real world applications. In these problems, agents have to attend to a stream of incoming tasks. Such Generalization of the Co-MAPF framework will bring the formulation closer to real-world problems. Several interesting questions arise in this context. For instance, how do we measure optimality and define the cost of meetings. Another issue relates to the fact that two collaborating agent finish their part in the task at different times. More specifically, the initiator agent may finish its part before the executor agent. We therefore need to reassign the initiator to another task (with another executor agent), and minimize the times agents are idle.

7.2.3 Task Assignment for Co-MAPF

Optimality guarantees and bounds. In Section 5.3 we suggested two approaches to integrate the `compute_assignments` algorithm into Co-CBS. The `compute_assignments` algorithm, presented in Section 5.1.3, finds approximate solutions to the S-D multi-index assignment problem, by relaxing the corresponding constrained optimization problem. We may wish to investigate how this affects the optimality guarantees of Co-CBS, and suggest ways to bound the deviation from the optimal cost in the solutions found by Co-CBS.

Approximating the global cooperation cost matrix. In our empirical evaluation, presented in Chapter 6, we examined the run time of the different algorithms for solving the Co-MAPF problem with task assignment. Specifically, we saw that for a large number of tasks, most of the time is spent on calculating the global cooperation cost matrix C. This matrix specifies the cost of completing task, for each combination of initiator agent and executor agent,

using the corresponding lowest-cost meeting. The time complexity of calculating C is of $\mathcal{O}(k^3)$, and we may suggest ways to make this more efficient by approximating the matrix C, possibly by eliminating combinations of agents and tasks, using some heuristic.

Task assignment effect on the success rate. Another phenomena we observed in the empirical evaluation, is the fact that assigning agents to tasks in an informed way can dramatically increase the success rate of the search algorithm. In other words, determining specific assignments has an impact on the hardness of the induced problem of finding conflict-free paths. This phenomena should be further investigated. This also relates to the issue of changing the layout of a warehouse, as discussed in [Salzman and Stern, 2020].

Appendix A

Planning for Cooperative Multiple Agents with Sparse Interaction Constraints

A.1 Introduction

The problem of cooperative multi-agent planning (MAP) is motivated by many real-world applications in a variety of domains, such as military, logistics, and search-and-rescue. In these problems, agents must coordinate their decisions to maximize their (joint) team value. When the state of the environment and all agents is fully-observable by each agent, the planning problem can be formalized as a multi-agent Markov decision process (MMDP, [Boutilier, 1996]). However, these models suffer from exponential increase in the size of the state and action spaces in the number of agents, which makes them computationally intractable in general. Specific structural assumptions are therefore required for an optimal solution to be feasible.

An important class of problems concerns high-level planning problems, where agents are essentially independent except for a prescribed set of possible interactions that can facilitate the plan execution. These types of problems are typically characterized by *loose coupling* and *sparse interactions* between agents, and some models exploit this fact to develop efficient algorithms. The complexity of such algorithms is often described by means of the problem coupling level. For instance, [Nissim et al., 2010] propose a fully distributed planning algorithm, based on the MA-STRIPS [Brafman and Domshlak, 2008] model, and [Melo and Veloso, 2011] propose approximate algorithms based on the decentralized sparse-interaction MDPs model.

Another common approach is to exploit the problem structure by using a compact representation with factored models. An example of such a representation is the *coordination graph* [Guestrin et al., 2002], also referred to as *interaction graph* [Nair et al., 2005] or *collaborative graphical games* [Oliehoek et al., 2012], which is solved using a graph-based optimization method, such as variable elimination (VE) [Guestrin et al., 2002; Larrosa and Dechter, 2003], or by distributed methods as investigated in the field of distributed constraint optimization problem (DCOPs, [Fioretto et al., 2018]).

It is also possible to exploit locality of interactions [Oliehoek et al., 2008; Melo and Veloso,

2011] and reward structure in transition-independent models, both centralized [Scharpff et al., 2016] and decentralized [Becker et al., 2004]. More specifically, in [Scharpff et al., 2016] reward dependencies are represented using conditional return graphs (CRGs) which are solved by a branch-and-bound policy search algorithm. In [Becker et al., 2004] a general formulation is suggested to represent the reward structure, using the notion of *events*. A coverage set algorithm is presented to find optimal policies. Scalability can often be improved even on more complex models, such as Network Distributed Partially Observable MDP (ND-POMDP), by leveraging sparse and structured interactions among agents. For example, the CBDP [Kumar and Zilberstein, 2009] algorithm is exponential only in the width of agents interaction graph.

In this paper, we focus on the multi-agent planning problem in a *deterministic* environment, where interactions between agents are symmetric and sparse. Possible interactions are captured using a notion of *soft cooperation constraints (SCC)*, where agents can affect the cost function by jointly satisfying prescribed constraints in state and time. This formulation is akin to the event-based formulation of [Becker et al., 2004], although less general to allow more specific and explicit computation schemes for each agent.

Based on the SCC model, we present a complete and optimal two-step planning algorithm, effective mostly in cases where interactions among agents are sparse. It is a dynamic programming (DP)-based algorithm, that decouples a multi-agent problem with K agents to K independent single-agent problems, such that the aggregation of the single-agent plans is optimal for the group. More specifically, in the first step we independently compute each agent's *response function*, which is its optimal plan with respect to all possible assignments of the timing variables of its associated constraints. We present an explicit algorithm for computing the response function, and provide a detailed complexity analysis. The second step is a centralized global *plan merging*, in which an optimal assignment to the timing variables is found under the *minimum-sum* objective. A *factor graph*, which captures dependencies among cooperative agents and exploits the internal structure of the problem, is applied to the problem with a variable elimination algorithm for efficient min-sum optimization.

Complexity analysis shows that the proposed algorithm is linear in the number of agents, polynomial in the span of the time horizon, and depends exponentially only on the number of interactions among agents.

We present a simulation implementing our proposed algorithm on a specific multi-agent planning problem. Our simulations show that the algorithm is efficient for this particular multi-agent setup and scales well in the number of agents compared to a standard solution.

We finally outline possible extensions to our model, to represent more complex cooperation constraints. For details of these extensions we refer [Revach, 2018].

The remainder of the paper is organized as follows. In section A.2 we present the model used and the formulation of SCC. Section A.3 presents a detailed description and implementation of our algorithm, followed by a complexity analysis. In section A.4 we present experimental results for our algorithm. In section A.5 we present an extension to our model to include asymmetric interactions between agents. Section A.6 concludes the paper and suggests directions for extensions and future research.

A.2 Model

We consider the finite horizon multi-agent deterministic planning problem. Our starting point is an MMDP with a factored state space, defined by a tuple $\langle \mathcal{T}, \mathbf{G}, \mathcal{S}, \mathcal{A}, \mathcal{H}, \mathcal{C}, \sigma_I, \sigma_* \rangle$, where

- $\mathcal{T} = \{0, ..., T\}$ is the time domain of length T.
- $\mathbf{G} = \{\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_K\}$ is a set of K agents.
- $\mathcal{S} = \mathcal{S}_1 \times ... \times \mathcal{S}_K$ is a finite state space, factored across agents, where $\mathcal{S}_k$ is the state space of agent $\mathbf{g}_k$.
- $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_K$ is a joint action space, similarly factored across agents.
- $\mathcal{H} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a deterministic transition function.
- $\mathcal{C} : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \cup \{\infty\}$ is a real-valued cost function.
- $\sigma_I \in \mathcal{S}$ is the initial state $\vec{s}_0$, and $\sigma_* \in \mathcal{S}$ the goal state.

Our objective is to find an optimal group policy $\vec{\pi}^*$ such that $\mathbf{J}^{\vec{\pi}}$ is minimal, i.e., $\vec{\pi}^* \in \arg\min_{\vec{\pi} \in \Pi^K} \mathbf{J}^{\vec{\pi}}$, where $\vec{\pi} = (\pi_1, ..., \pi_K)$ is the joint policy, $\mathbf{J}^{\vec{\pi}}$ is the aggregate cumulative cost defined by

$$\mathbf{J}^{\vec{\pi}} = \sum_{t=0}^{T-1} \mathcal{C}\left(\vec{s}_t, \vec{a}_t\right) \tag{A.1}$$

if $\vec{s}_T = \sigma_*$, and $\mathbf{J}^{\vec{\pi}} = \infty$ otherwise. Here $\vec{a}_t = (a_{t,1}, a_{t,2}, ..., a_{t,K}) \in \mathcal{A}$ is the joint action at time t, such that $a_{t,k} = \pi_k\left(s_t, t\right)$.

We next describe the sparse interactions structure. We first assume transition and cost independence across agents, namely

$$\mathcal{H}\left(\vec{s}, \vec{a}\right) = \left(\mathcal{H}_1\left(s_1, a_1\right), \ldots, \mathcal{H}_K\left(s_K, a_K\right)\right) \tag{A.2}$$

and

$$\mathcal{C}\left(\vec{s}, \vec{a}\right) = \sum_{k=1}^{K} \mathcal{C}_k\left(s_k, a_k\right) \tag{A.3}$$

Coupling between agents is introduced via a set $\Psi = \{\psi_1, ..., \psi_L\}$ of *soft cooperation constraints*. Each constraint ψ_ℓ defines a single opportunity for cooperative interaction between agents. In particular, a constraint ψ_ℓ, $\ell \in \{1, ..., L\}$, is specified by the following tuple:

$$\psi_\ell = \left\langle \mathbf{G}_\ell, \Sigma_\ell, \mathcal{C}_\ell^-, \mathcal{T}_\ell \right\rangle \tag{A.4}$$

where

- $\mathbf{G}_\ell = \{\mathbf{g}_k,\ k \in K_\ell\}$, with $K_\ell = (k_{\ell,1}, \ldots, k_{\ell,n(\ell)})$, is the set of $n(\ell)$ agents interacting in constraint ψ_ℓ.
- $\Sigma_\ell = \{\sigma_{\ell,k}, k \in K_\ell\}$, with $\sigma_{\ell,k} \in \mathcal{S}_k$, is a set of local interaction states. Namely, for the constraint to hold, agent k is required to be in state $\sigma_{\ell,k}$ at some prescribed time.

- $\mathcal{C}_\ell^-$ is a (reduced) immediate cost for the group for interaction, applicable when the constraint is satisfied (see equation A.5).

- $\mathcal{T}_\ell$ is the constraint time domain; i.e., it is a subset of time instances at which the interaction may take place: $\mathcal{T}_\ell \subseteq \{0, 1, ..., T-1\} \cup T_\emptyset$. Here $T_\emptyset$ is a special notation for the *null assignment*, where the constraint is not satisfied, i.e., there is no interaction.

Note that the SCC formulation can be extended to represent more general constraints. For instance, a constraint can have a set of time domains, one for each agent, such that each agent interacts at a different time. Moreover, a constraint can have a subset of interaction states (instead of a single state). While the ideas are similar, for concreteness and brevity we leave these extensions to future work.

A.2.1 Interaction-Dependent Cost

Agents are coupled only via the constraint set Ψ. Therefore, the group cost depends on the constraints satisfied, where each satisfied constraint ψ_ℓ represents an interaction which applies the group a reduced cost $\mathcal{C}_\ell^-$. We now describe the structure of the group cost under this formulation.

Let $\tau_\ell \in \mathcal{T}_\ell$ be an *interaction timing variable* that defines the timing of the interaction under constraint ψ_ℓ. For a given assignment to the timing variable τ_ℓ, we define an indicator function that is true if all interacting agents in $\mathbf{G}_\ell$ satisfy constraint ψ_ℓ:

$$\hat{\psi}_\ell\left(\tau_\ell; \vec{\pi}\right) = \mathbb{I}_{\{\tau_\ell \neq T_\emptyset\}} \prod_{k \in K_\ell} \mathbb{I}_{\{s_{\tau_\ell, k} = \sigma_{\ell,k}\}} \tag{A.5}$$

where $\mathbb{I}_A$ is the 0/1 indicator of event A. Namely, constraint ψ_ℓ is satisfied given $\tau_\ell = \tau$ if $\tau \neq T_\emptyset$ and all interacting agents in ψ_ℓ arrive at their interaction state at time τ.

Furthermore, $\vec{\tau}$ is the interaction vector, and $\mathcal{D}$ is its domain, i.e., the cross space of all constraint time domains:

$$\vec{\tau} = (\tau_1, \tau_2, ..., \tau_L) \in \mathcal{T}_1 \times \mathcal{T}_2 \times ... \times \mathcal{T}_L \triangleq \mathcal{D} \tag{A.6}$$

$\vec{\tau}_k$ is the timing vector of all constraints involving agent $\mathbf{g}_k$ (with domain $\mathcal{D}_k$).

Under this new formulation, given an initial state $\sigma_I \in \mathcal{S}$, a goal state $\sigma_* \in \mathcal{S}$, and a constraint set Ψ, our objective is to find the optimal group policy where $\mathbf{J}^{\vec{\pi}}$ in equation A.1 is now

$$\mathbf{J}^{\vec{\pi}}\left(\vec{\tau}\right) = \sum_{t=0}^{T-1} \sum_{k=1}^{K} \mathcal{C}_{0,k}\left(s_{t,k}, a_{t,k}\right) + \sum_{\ell=1}^{L} \hat{\psi}_\ell\left(\tau_\ell; \vec{\pi}\right) \mathcal{C}_\ell^- \tag{A.7}$$

where $\mathcal{C}_{0,k}$ is the single-agent independent immediate cost with no consideration of interactions. Namely, it is the sum of all agents' independent immediate cost plus the sum of the reduced costs of all satisfied constraints.

Note that now the multi-agent optimal policy $\vec{\pi}$ is a parametric policy with respect to timing variables, and the aggregate cumulative cost $\mathbf{J}^{\vec{\pi}}$ is a function of the timing variables. Effectively, there may be a different optimal policy for each assignment of timing variables. Furthermore,

$\mathbf{J}_k^*(\vec{\tau}_k)$ is the optimal response function (i.e., the optimal cumulative cost) for agent $\mathbf{g}_k$ given an assignment of the timing vector $\vec{\tau}$.

Our objective is to minimize the multi-agent cumulative cost under L interaction constraints:

$$\mathbf{J}^* = \min_{\vec{\tau}\in\mathcal{D}} \min_{\vec{\pi}\in\Pi^K} \left\{ \mathbf{J}^{\vec{\pi}}(\vec{\tau}) \right\} \tag{A.8}$$

where $\mathbf{J}^{\vec{\pi}}(\vec{\tau})$, defined by equation A.7, is decomposable, and where each single agent cost function depends only on the single agent policy. Therefore, we may switch the order of summation to compute independently for each agent:

$$\mathbf{J}^{\vec{\pi}}(\vec{\tau}) = \sum_{k=1}^{K} \sum_{t=0}^{T-1} \mathcal{C}_{0,k}(s_{t,k}, a_{t,k}) + \sum_{\ell=1}^{L} \hat{\psi}_\ell(\tau_\ell; \vec{\pi})\, \mathcal{C}_\ell^- \tag{A.9}$$

provided that $\vec{s}_T = \sigma_*$ and $\mathbf{J}^{\vec{\pi}}(\vec{\tau}) = \infty$ otherwise.

We can then minimize each single agent cost independently for any given assignment of the timing vector $\vec{\tau} \in \mathcal{D}$ (and specifically $\vec{\tau}_k$ for each agent $\mathbf{g}_k$). After the optimal single agent response functions are found, we need to find the optimal assignment for the timing variables. Let us observe that the multi-agent problem decomposition results in a min-sum optimization problem:

$$\vec{\tau}^* \in \arg\min_{\vec{\tau}\in\mathcal{D}} \sum_{k=1}^{K} \mathbf{J}_k^*(\vec{\tau}_k) \tag{A.10}$$

that is, the sum of optimal response functions. We can use this structure to our advantage by applying an efficient optimization algorithm.

A.3 *DIPLOMA* - Distributed Planning and Optimization Algorithm for Multiple Agents

In this section we present the *DIstributed PLanning and Optimization algorithm for Multiple Agents (DIPLOMA)*, which addresses the previous multi-agent interaction model and optimizes cost and policy. Using this model, we are able to decompose a global multi-agent planning problem into a two-step problem. First, K distributed independent single-agent planning problems are solved. Second, we optimize the global solution with respect to the cooperation constraints by selecting a plan for each agent.

We now describe the steps of our proposed algorithm, presented in algorithm A.1:

1. **Response Function Computation**

   For every agent $\mathbf{g}_k \in \mathbf{G}$, compute the single agent response function independently,

   $$\forall \vec{\tau}_k \in \mathcal{D}_k \ , \ \mathbf{J}_k^*(\vec{\tau}_k) = \min_{\pi_k\in\Pi_k} \mathbf{J}_k^{\pi_k}(\vec{\tau}_k) \tag{A.11}$$

   It may be computed using various dynamic programming algorithms, and more specifically using the algorithms described next, in detail. This step can be parallelized over agents.

2. **Plan Merging**

   Compute the optimal total multi-agent cost by minimizing the sum single agent response with respect to the constraint variables. More specifically:

$$\mathbf{J}^* = \min_{\vec{\tau} \in \mathcal{D}} \sum_{k=1}^{K} \mathbf{J}_k^* \left(\vec{\tau}_k \right) \tag{A.12}$$

   The minimization process can be carried out efficiently using factor graph modeling and a variable elimination algorithm, as described below. Let $\vec{\tau}^*$ denote the optimal assignment of the constraint variables.

3. **Policy Backtracking**

   (a) For every agent $\mathbf{g}_k \in \mathbf{G}$, backtrack the single agent optimal policy independently:

$$\pi_k^* \in \arg\min_{\pi_k \in \Pi_k} \mathbf{J}_k^{\pi_k} \left(\vec{\tau}_k^* \right) \tag{A.13}$$

   (b) The global optimal multi-agent policy is then given by

$$\vec{\pi}^* = \{ \pi_1^*, \pi_2^*, ..., \pi_k^*, ..., \pi_K^* \} \tag{A.14}$$

Algorithm A.1 DIPLOMA

returns $\vec{\pi}^*$, the optimal group policy
inputs: MMDP, Ψ
for all $\mathbf{g}_k \in \mathbf{G}$ **do** ▷ response function computation
    for all $\vec{\tau}_k \in \mathcal{D}_k$ **do**
        $\mathbf{J}_k^* \left(\vec{\tau}_k \right) = \min_{\pi_k \in \Pi_k} \mathbf{J}_k^{\pi_k} \left(\vec{\tau}_k \right)$
$\mathbf{J}^* = \min_{\vec{\tau} \in \mathcal{D}} \sum_{k=1}^{K} \mathbf{J}_k^* \left(\vec{\tau}_k \right)$ ▷ plan merging
$\vec{\tau}^* \in \arg\min_{\vec{\tau} \in \mathcal{D}} \sum_{k=1}^{K} \mathbf{J}_k^* \left(\vec{\tau}_k \right)$
for all $\mathbf{g}_k \in \mathbf{G}$ **do**
    $\pi_k^* \in \arg\min_{\pi_k \in \Pi_k} \mathbf{J}_k^{\pi_k} \left(\vec{\tau}_k^* \right)$
$\vec{\pi}^* = \{ \pi_1^*, \pi_2^*, ..., \pi_k^*, ..., \pi_K^* \}$
return $\vec{\pi}^*$

A.3.1 Response Function Computation

The main step of our proposed algorithm is computing the single agent response function with respect to constraint timing variables. We now describe algorithms to compute $\mathbf{J}_k^*$ efficiently for each agent. Before describing the algorithms in detail, we present a few basic definitions and notations:

- The algorithms presented are from a single agent perspective; therefore, we omit the index k from the notation wherever possible.

- $\mathcal{V}^*(\sigma, \sigma_*, \tau)$, the *cost-to-state*, is the optimal cumulative cost from state σ at time $t = \tau$ to the target state σ_* in $T - \tau$ time steps.
- $\mathcal{J}^*(\sigma_I, \sigma, \tau)$, the *cost-from-state*, is the optimal cumulative cost from initial state σ_I at time $t = 0$ to state σ in τ time steps, and $\mathbf{J}^* = \mathcal{J}^*(\sigma_I, \sigma_*, T)$.
- More generally, $\mathcal{V}^*_{\Diamond}(s_I, s_g, \tau_I, \tau_g)$ is the optimal cumulative cost from state s_I at time $t = \tau_I$ to state s_g at time $t = \tau_g$ in $\tau_g - \tau_I$ time steps.

We start with the following computation in algorithm A.2 of the cost-to-state and cost-from-state. The result is required only for intermediate states in the agent's constraint set. A natural implementation by Dynamic Programming (DP) computes these costs via a single pass for all states and times.

Algorithm A.2 Costs to and from states

1: **for all** $\tau \in \{1, .., T\}$ and $\sigma \in \{\sigma_\ell\}$ **do**
2: Compute $\mathcal{J}^*(\sigma_I, \sigma, \tau)$ iteratively using DP.
3: **for all** $\tau \in \{T-1, .., 0\}$ $\sigma \in \{\sigma_\ell\}$ **do**
4: Compute $\mathcal{V}^*(\sigma, \sigma_*, \tau)$ iteratively using DP.
5: Cache all results for later use.

The single agent response function is the optimal cumulative cost with respect to the timing variables, i.e., the optimal plan from the initial state to the goal state, while satisfying the constraints in times specified by the timing variables. To simplify the presentation, we start by showing how to compute the single-agent response function with a single interaction (i.e., a single constraint), and then follow with the general case of L interactions.

Let τ_ℓ be a single timing variable (i.e., $L = \ell = 1$), and $\mathbf{J}^*_\sigma(\tau_\ell)$ the optimal cumulative cost from initial state σ_I to goal state σ_* in T time steps, via the intermediate state σ_ℓ; i.e., $s_{\tau_\ell} = \sigma_\ell$. For a given assignment of τ_ℓ, we can compute the response function in this simple case as:

$$\mathbf{J}^*_\sigma(\tau) = \begin{cases} \mathbf{J}^*, & \tau = T_\emptyset \\ \mathcal{J}^*(\sigma_I, \sigma_\ell, \tau) + \mathcal{V}^*(\sigma_\ell, \sigma_*, \tau), & \text{otherwise} \end{cases} \tag{A.15}$$

Namely, it is computed by two parts: planning from the initial state σ_I in time $t = 0$ to the constraint state σ_ℓ in time $t = \tau_\ell$, and from the latter to the goal state at T (i.e., for $T - \tau$ time steps). If $\tau_\ell = T_\emptyset$, the constraint is not imposed. Hence, $\mathbf{J}^*_\sigma(\tau_\ell) = \mathbf{J}^*$, i.e., the optimal cost with no consideration of interactions.

The generalization for $L = L_k$ constraints (the number of constraints agent k is involved in) follows the same idea. We need to compute the response function $\mathbf{J}^*_{\sigma_1,...,\sigma_L}(\tau_1, ..., \tau_L)$ for every assignment of L timing variables. We present an incremental scheme that efficiently avoids repeated computation of given segments:

1. Pre-compute the state-to-state cost functions by dynamic programming, and cache the results for later use:
   1.1. Apply algorithm A.2.

1.2. Pre-compute $\mathcal{V}^*_\Diamond$ using algorithm A.3.

2. Build the response function from the bottom up using the previously cached values that were pre-computed in the previous step, using algorithms A.4 and A.5. For simplicity we use the following concise notation, for $1 \leq \ell \leq L$:

$$\mathbf{J}^* \{\ell\} \triangleq \mathbf{J}^*_{\sigma_1,...,\sigma_\ell} (\tau_1, ..., \tau_\ell) \tag{A.16}$$

In algorithm A.5 we show how to compute $\mathbf{J}^* \{\ell + 1\}$ for all assignments to $\tau_{\ell+1}$, given $\mathbf{J}^* \{\ell\}$ for a specific assignment to $\tau_1, \ldots, \tau_\ell$. The idea is essentially to replace $\mathcal{V}^*_\Diamond (\sigma_{\ell_1}, \sigma_{\ell_2}, \tau_{\ell_1}, \tau_{\ell_2})$ by the sum $\mathcal{V}^*_\Diamond (\sigma_{\ell_1}, \sigma_{\ell+1}, \tau_{\ell_1}, \tau_{\ell+1}) + \mathcal{V}^*_\Diamond (\sigma_{\ell+1}, \sigma_{\ell_2}, \tau_{\ell+1}, \tau_{\ell_2})$ when constraint $\ell + 1$ is added with timing assignment $\tau_{\ell+1}$ between existing τ_{ℓ_1} and τ_{ℓ_2}.

Algorithm A.3 Multiple constraint response - step 1.2

```
for all σ_i, σ_j ∈ {σ_1, σ_2, ..., σ_L} do
    for all τ_i ∈ T_i do
        for all τ_j ∈ T_j, τ_j > τ_i do
            Compute V*_◇ (σ_i, σ_j, τ_i, τ_j)
            Cache the results for later use.
```

Algorithm A.4 Multiple constraint response - step 2

```
for all ℓ ∈ {1, 2, ..., L − 1} do
    for all τ_1, τ_2, ..., τ_ℓ do
        For a given assignment to τ_1, τ_2, ..., τ_ℓ
        Such that τ_{i_1} ≤ τ_{i_2} ≤ ... ≤ τ_{i_ℓ}
        Compute J* {ℓ + 1} from J* {ℓ} for all τ_{ℓ+1} ∈ T_{ℓ+1}, using Algorithm A.5
```

A.3.2 Plan Merging

The plan merging step of our proposed algorithm requires finding an optimal assignment to the timing variables while optimizing the global cost function $\mathbf{J}^* (\vec{\tau})$. This is a weighted constraint satisfaction programming problem, which is $\mathcal{NP}$-hard in the general case [Larrosa and Dechter, 2003]. In the special case of the min-sum objective (equation A.12), we can reduce optimization complexity by using models that consider the internal structure of the dependency among agents. A graphical model, called factor graph [Loeliger, 2004], describes the interaction among agents, and captures agent dependency or independency, therefore leading to more efficient optimization algorithms.

A factor graph contains variable nodes representing constraint variables (the timing variables), and factor nodes representing single-agent cost functions $\mathbf{J}^*_k (\vec{\tau}_k)$. Edges connect a cost function to all the variables associated with the constraints involved in that cost function. Figure A.1 illustrates how the min-sum optimization problem is represented using a factor graph. In this example, we have three cooperation constraints, where $\mathbf{G}_1 = \{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3\}$, $\mathbf{G}_2 = \{\mathbf{g}_1, \mathbf{g}_3\}$, and $\mathbf{G}_3 = \{\mathbf{g}_3, \mathbf{g}_4\}$.

Algorithm A.5 Multiple constraint response - inner algorithm

Assume $\tau_{i_1} \leq \tau_{i_2} \leq ... \leq \tau_{i_\ell}$
$p = 1$ ▷ Initialize pivot index
$b = 1$ ▷ Set *baseline* flag
for all $\tau_{\ell+1} \in \{0, 1, ..., T-1\}$ **do**
  if $p = 1$ **then**
    if $b = 1$ **then**
      $\mathbf{J}^*_{base} = \mathbf{J}^*\{\ell\} - \mathcal{J}^*(\sigma_I, \sigma_{i_1}, \tau_{i_1})$ ▷ Initialize a *baseline* value for $\mathbf{J}^*\{\ell+1\}$
      $b = 0$ ▷ Reset *baseline* flag
    if $\tau_{\ell+1} < \tau_{i_1}$ **then**
      $\mathbf{J}^*\{\ell+1\} = \mathbf{J}^*_{base} + \mathcal{J}^*(\sigma_I, \sigma_{\ell+1}, \tau_{\ell+1}) + \mathcal{V}^*_{\Diamond}(\sigma_{\ell+1}, \sigma_{i_1}, \tau_{\ell+1}, \tau_{i_1})$
    else ▷ $\tau_{\ell+1} = \tau_{i_1}$
      $\mathbf{J}^*\{\ell+1\} = \infty$ ▷ There is no valid plan that meets the constraints
      $p = 2$ ▷ Increment pivot index
      $b = 1$ ▷ Set *baseline* flag
  else if $1 < p \leq \ell$ **then**
    if $b = 1$ **then**
      $\mathbf{J}^*_{base} = \mathbf{J}^*\{\ell\} - \mathcal{V}^*_{\Diamond}(\sigma_{i_{p-1}}, \sigma_{i_p}, \tau_{i_{p-1}}, \tau_{i_p})$
      $b = 0$ ▷ Reset *baseline* flag
    if $\tau_{\ell+1} < \tau_{i_p}$ **then**
      $\mathbf{J}^*\{\ell+1\} = \mathbf{J}^*_{base} + \mathcal{V}^*_{\Diamond}(\sigma_{i_{p-1}}, \sigma_{\ell+1}, \tau_{i_{p-1}}, \tau_{\ell+1}) + \mathcal{V}^*_{\Diamond}(\sigma_{\ell+1}, \sigma_{i_p}, \tau_{\ell+1}, \tau_{i_p})$
    else ▷ $\tau_{\ell+1} = \tau_{i_p}$
      $\mathbf{J}^*\{\ell+1\} = \infty$ ▷ There is no valid plan that meets the constraints
      $p = p + 1$ ▷ Increment pivot index
      $b = 1$ ▷ Set *baseline* flag
  else ▷ $p > \ell$
    if $b = 1$ **then**
      $\mathbf{J}^*_{base} = \mathbf{J}^*\{\ell\} - \mathcal{V}^*(\sigma_{i_\ell}, \sigma_*, \tau_{i_\ell})$ ▷ Initialize a *baseline* value for $\mathbf{J}^*\{\ell+1\}$
      $b = 0$ ▷ Reset *baseline* flag
    $\mathbf{J}^*\{\ell+1\} = \mathbf{J}^*_{base} + \mathcal{V}^*_{\Diamond}(\sigma_{i_\ell}, \sigma_{\ell+1}, \tau_{i_\ell}, \tau_{\ell+1}) + \mathcal{V}^*(\sigma_{i_{\ell+1}}, \sigma_*, \tau_{i_{\ell+1}})$
$\mathbf{J}^*\{\ell+1\}(T_\emptyset) = \mathbf{J}^*\{\ell\}$ ▷ $\tau_{\ell+1}$ is equal to the *null* assignment, namely no constraint

On the factor graph we apply a variable elimination (VE) algorithm, which is used mainly for exact inference [Koller and Friedman, 2009]. VE exploits the internal structure of the problem and reduces computations [Larrosa and Dechter, 2003].

The factor graph structure and the VE elimination ordering have a major effect on the complexity and efficiency of the algorithm, which is out of the scope of this work (see [Koller and Friedman, 2009]). However, in the next section we present several representative cases. The scheme for applying VE to solve the optimization problem is described in [Revach, 2018].

A.3.3 Complexity Analysis

In this section we present an overall complexity analysis of our proposed algorithm. We first present the complexity of the response function computation, followed by an overall analysis of a few representative cases, and establish an upper bound on the complexity of planning problems. The complexity result is formulated in terms of the overhead of planning for a multi-agent system as a function of planning for each single agent in isolation, when considering the same problem structure. More specifically, we denote $\mathbf{T}(\mathcal{V}^*, T)$ and $\mathbf{T}(\mathcal{J}^*, T)$ as the time complexity

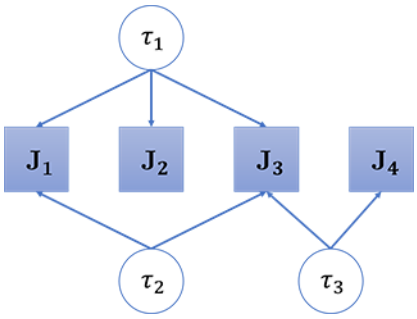


Figure A.1: Factor graph example.

of computing a single-agent cost-to-state and cost-from-state, respectively, over the time horizon T, and assume $\mathbf{T}(\mathcal{V}^*, T) = \mathbf{T}(\mathcal{J}^*, T)$.

For the response function computation, the first step of pre-computation (algorithms A.2 and A.3) is of the order of $L^2 \cdot \sum_{\tau_\ell \in \mathcal{T}_\ell} \mathbf{T}(\mathcal{V}^*, \tau_\ell) \leq L^2 \cdot \frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*, T)$, where L is the number of constraints in which the agent is involved. We can use an efficient algorithm for computing a single agent cost-to-state from every initial state to a fixed and specific goal state (e.g., using dynamic programming) and denote its time complexity as $\mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T)$. In that case, we may reduce the time complexity by a factor of L, compared to single agent planning, i.e., $L \cdot \frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T)$. The time complexity of the second step (algorithm A.4) is dominated by $\mathcal{O}\left(T^L\right)$. Therefore, the overall complexity for computing the response function is

$$L \cdot \frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T) + \mathcal{O}\left(T^L\right) \tag{A.17}$$

In the case of a single constraint, this reduces to $2 \cdot \mathbf{T}(\mathcal{V}^*, T) + \mathcal{O}(T)$ (equation A.15).

The complexity of the plan merging step, and more specifically the VE algorithm, depends on the scope size of each factor; that is, the number of variables to which each factor is connected. The total complexity has an exponential dependency in the scope size of the factors and it is of the order of $\mathcal{O}((K+L) \cdot d^m)$ where m is the maximal scope size of factors and d is the maximal number of values of each variable. For a detailed complexity analysis of the VE algorithm on a factor graph, see [Revach, 2018; Koller and Friedman, 2009].

We now present an analysis of the overall time complexity for several representative cases. We start with a very sparse case, where there are $2 \cdot L$ agents, each of which is involved in only one cooperation constraint, i.e., $K = 2 \cdot L$. The response function computation time complexity is dominated by $2 \cdot \mathbf{T}(\mathcal{V}^*, T) + \mathcal{O}(T)$ and is linear in the span of the time horizon. Each timing variable does not depend on any of the other variables. The time complexity of eliminating a single variable is dominated by $\mathcal{O}(T)$; i.e., it is also linear in the span of the time horizon. The overall complexity is $K \cdot [2 \cdot \mathbf{T}(\mathcal{V}^*, T) + \mathcal{O}(T)] + L \cdot \mathcal{O}(T) = 2 \cdot K \cdot \mathbf{T}(\mathcal{V}^*, T) + \frac{3}{2} \cdot K \cdot \mathcal{O}(T)$. Because of the inherent decoupling in this case, this is equal to solving $L = \frac{K}{2}$ independent problems.

In a dense case we consider two agents with $L \geq 2$ cooperation constraints between them (i.e., each agent is involved in L constraints). The time complexity of the response function computation is exponential in L. As all the timing variables belong to the same factors, they are therefore dependent. The time complexity of the plan merging is also exponential in L, but it is not the dominating part. The overall complexity is dominated by $2 \cdot L \cdot \left(\frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T) + \mathcal{O}\left(T^L\right)\right)$.

We now consider an hierarchical case, where each constraint involves two agents and the factor graph is a balanced N-tree with depth M. There are N^M agents (factors) that are represented as leaf nodes in the tree, and $K - N^M$ agents that are not represented as leaf nodes. Here, K is equal to $K = \sum_{m=0}^{M} N^m$; therefore, the total number of cooperation constraints is equal to $L = K - 1$. Each of the leaf agents is involved in only one cooperation constraint; therefore, the complexity of computing their response function is just linear: $N^M \cdot (2 \cdot \mathbf{T}(\mathcal{V}^*, T) + \mathcal{O}(T))$. An agent that is not a leaf node in the tree is involved in $N+1$ cooperation constraints. Therefore, the complexity of computing their response function is $\left(K - N^M\right) \cdot (N+1) \cdot \left(\frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T) + \mathcal{O}\left(T^{N+1}\right)\right)$. In the case of a tree, the plan merging is executed bottom up from the leaf nodes to the root node. Every factor that is not a leaf generates an $N+1$ cliques (see [Koller and Friedman, 2009]) of timing variables (i.e., all the variables on which the factor depends). Therefore, the complexity of plan merging is dominated by the size and number of cliques. The complexity of eliminating a clique by a VE algorithm is dominated by $\mathcal{O}\left(T^{N+1}\right)$, and the number of cliques is equal to $\mathbf{C_L} = K - N^M$. Note that the process of eliminating cliques in the same level of the tree can be distributed and parallelized.

Finally, we define a coupling measure ρ for the system as the maximal number of constraints in which each agent is involved,

$$\rho = \max_k L_k, \quad k = 1, \ldots, K \tag{A.18}$$

where L_k is the number of constraints in which agent $\mathbf{g}_k$ is involved. An upper bound for the complexity is linear in K, polynomial in T, and exponential only in ρ:

$$\mathcal{O}\left(K \cdot \rho \cdot \left(\frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T) + T^{\rho}\right)\right) \tag{A.19}$$

A.4 Experiments

In this section we present the results of basic experiments performed using DIPLOMA, in order to validate its correctness and test its time complexity. We compare the algorithm's performance against a centralized DP algorithm solving the underlying MMDP. We use the same DP algorithm for calculating $\mathcal{J}^*(\sigma_I, \sigma, \tau)$ and $\mathcal{V}^*(\sigma, \sigma_*, \tau)$ in algorithm A.2. All simulations were performed on an Intel i7-8700 CPU @ 3.20Ghz machine with 16.0 GB RAM.

We ran our simulations on a simple grid world example where several agents have to travel from an initial location to a goal location in T time steps while collecting as many boxes as possible. Each box has its own reward and associated agent, and some boxes can be picked by two agents together in order to gain a double reward. In order for agents to pick a box together, they have to meet at the box location at the same time. Agents can move *up*, *left*, or *right*, and collect the reward upon moving up from their box location. Our goal is to find an optimal joint plan such that the group reward is maximized. Note that in this example we use reward instead of cost used in the model; however, replacing between the two is trivial by taking negative rewards. This problem is depicted in figure A.2 for a grid of 10×10 and four agents ($K = 4$). This problem is quite simple but can represent scheduling problems, box-pushing, search-and-rescue and more.

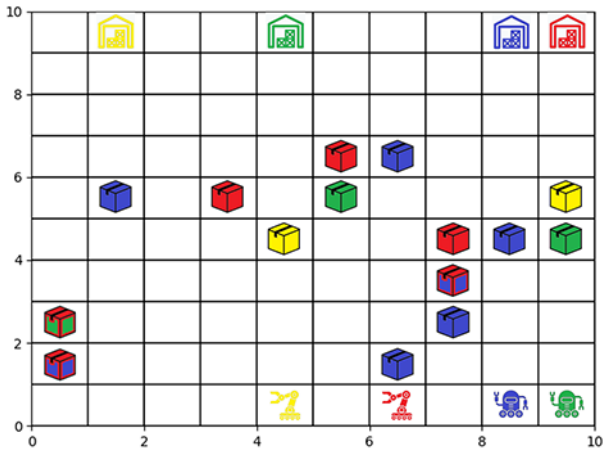


Figure A.2: A box-collecting problem on a 10×10 grid with four agents ($K = 4$), denoted by four different colors. All agents start in the bottom row and have to arrive to the corresponding warehouse at the top row within T time steps. Each agent can only pick boxes of its color. Agents can cooperate in three different locations ($L = 3$), illustrated by a two-colored box. For instance, in the box located in $(0, 2)$, the red agent can pick the box alone and receive a reward of 3, or it can pick it with the assistance of the blue agent and receive a reward of 6.

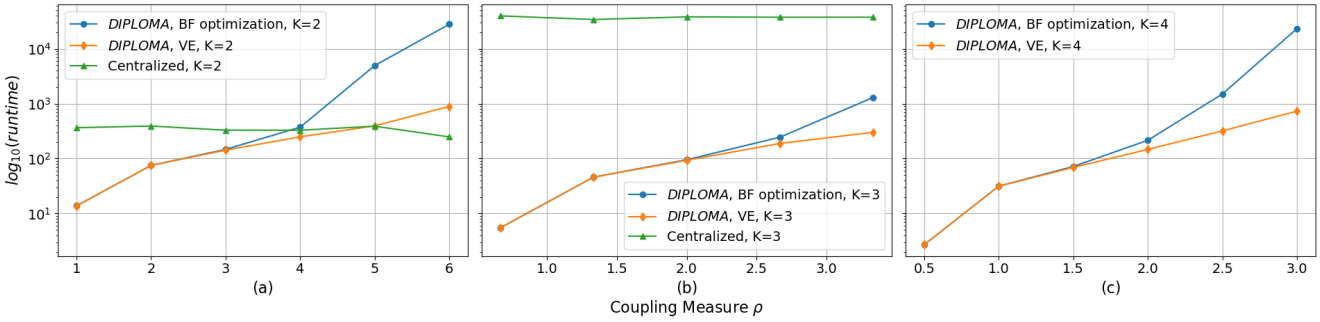


Figure A.3: Simulation results for $K = 2$ (a), $K = 3$ (b), and $K = 4$ (c) on a logarithmic scale. The centralized reference planner does not depend on the coupling measure (i.e., number of constraints in the problem), but scales poorly on the number of agents, and for $K = 4$ is practically infeasible. DIPLOMA algorithm achieves an improvement of 2 orders of magnitude, and depends on the coupling level.

We ran our simulation on a fixed grid size of 10×10 $\left(|\mathcal{S}| = 100^K\right)$, a fixed horizon of $T = 20$ time steps, and different values for number of agents (K), and constraints (L). For every value of K we generated 20 random environments (with a random number of constraints) and measured the runtime of the centralized reference algorithm and DIPLOMA. Figure A.3 demonstrates how our proposed algorithm scales in the number of agents, and depends on the coupling measure ρ (see equation A.18). We also compare the runtime of our algorithm using the VE algorithm for the plan merging step, compared to a brute-force (BF) optimization. Elimination ordering for VE was determined by a simplified min-neighbors criteria [Koller and Friedman, 2009]. The reference algorithm does not depend on the coupling measure (i.e., number of constraints), but for $K = 3$, has a runtime higher by two orders of magnitude than DIPLOMA. A value of $K = 4$ makes it practically infeasible to run. DIPLOMA, on the other hand, scales well in the number of agents and depends mostly on the coupling level. Furthermore, using VE optimization for the plan merging step (compared to a brute-force optimization), reduces runtime significantly when the coupling measure increases.

A.5 Extension to Asymmetric Interactions

In this paper we focus on simple symmetric interactions between agents, i.e., meeting constraints where all agents must arrive at the same time for the group to benefit from the interaction. Our model, however, can be extended to include asymmetric and more complex temporal constraints, enabling a compact representation of such constraints. Furthermore, it enables the development of efficient planning algorithms that exploit the linear time complexity of solving an MDP. This can be done by applying the group interaction cost $\mathcal{C}_\ell^-$ to a specific interacting agent, called the *affected* agent, and embedding an *activation function* of the form $f_\ell : \mathcal{T} \times \mathcal{T}_\ell \to \{0, 1\}$ into the affected agent's immediate cost. The activation function defines a set of time instances where the interaction cost is applicable.

As an example, one can consider a scenario where a *facilitating* agent can arrive at a certain state in time $\tau \in \mathcal{T} = \{1, ..., 10\}$, which opens a 10 time steps window following time τ, allowing the second agent to receive an additional reward for each time step (within this time window) in which it is in a related state. If we formulate this interaction using a distinct constraint for each possible state-time pair, we need $10^2 = 100$ constraints, and thus checking about $2^{100} \approx 10^{30}$ different combinations of constraints. By formulating this interaction with an SCC, using a step activation function, we would have only 10 constraints. We would need to check only $2^{10} = 1024 \approx 10^3$ combinations of constraints of the first agent, and for each one, solve a single induced MDP for the second agent. Thus, we obtain an improvement of many orders of magnitude in this simple case.

A detailed formulation and implementation of this extension is presented in [Revach, 2018], including an efficient asymmetric planning algorithm using a step activation function.

Another rather trivial extension to asymmetric interactions is to use a different interaction timing for every interacting agent in a constraint. Since we calculate the agents' response for each $\tau \in \{0, \ldots, \mathcal{T}\}$ (see algorithm A.5), we can choose a different value of τ for each agent in the plan merging step.

A.6 Discussion and Future Work

In this paper, we address the problem of fully cooperative multiple agents high-level planning problems in deterministic environments. We focus on problems where interactions between agents are symmetric and sparse, and present a framework for representing all interactions as *soft cooperation constraints* (SCC). This framework enables a compact representation of temporal constraints and can be further extended and generalized to include more types of constraints. Considering the SCC formulation, only those agents that are subject to the same cooperation constraint are coupled, forming a dependency only in a specific context.

The SCC model presented is quite general and useful in practice, and can express constraints used in realistic scenarios. The main use case is coordination of high-level actions among autonomous agents. Example problems are the coordination of rescue or military forces, the Mars rover exploration (discussed in [Becker et al., 2004]) or the coordinated target tracking (discussed in [Kumar and Zilberstein, 2009]). We can extend several combinatorial optimization problems, such as the vehicle routing problem (VRP) or the multiple traveling salesman problem, to include potential meetings between agents that provide additional rewards for the group. An SCC can also express a conflict (or collision) constraint (specifically in multi-agent path finding problems) by setting a positive or infinite interaction cost (see section A.2), and using the extended formulation presented in section A.5. In a similar way, we can also represent resource constraints, such as "use at most 1 of this resource at the same time", by adding constraints to states where the resource is used by agents.

Using this model, we are able to describe an efficient algorithm, *DIPLOMA*, which is both complete and optimal. The proposed algorithm is a two-step algorithm: a dynamic programming-based planning step and an optimization step.

In the first step, each agent plans independently and computes its response function to the associated constraints with respect to interaction timing variables. We show non-trivial and efficient algorithms for computation, which can also be distributed and parallelized. The time complexity per agent strongly depends on the span of the time horizon and the number of cooperation constraints relevant to this particular agent.

In the second step, we use a variable elimination algorithm on a factor graph to find the optimal assignment to timing variables. The algorithm exploits the internal structure of the problem and independence among agents to efficiently solve the min-sum optimization problem.

A theoretical time complexity analysis is presented, showing that the overall algorithm is linear rather than exponential in the number of agents, polynomial in the span of the time horizon, and dependent on the number of interactions among agents.

Simulations show that the algorithm is efficient compared to a standard solution and scales well in the number of agents.

An immediate direction for future research is the extension to more expressive interaction constraints, as discussed in section A.5. Other possible directions for future research include generalizing the formulation of constraints by expanding the state and time domains of each constraint, defining types of agents (rather than specific agents) in a constraint, approximate methods for computing the response functions, and simulating a real-world large-scale MAP problem.

Bibliography

Atzmon, D., Freiman, S. I., Epshtein, O., Shichman, O., and Felner, A. (2021). Conflict-free multi-agent meeting. In *Int. Conf. Automated Planning and Scheduling (ICAPS)*, pages 16–24.

Atzmon, D., Li, J., Felner, A., Nachmani, E., Shperberg, S. S., Sturtevant, N., and Koenig, S. (2020a). Multi-directional heuristic search. In *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 4062–4068.

Atzmon, D., Zax, Y., Kivity, E., Avitan, L., Morag, J., and Felner, A. (2020b). Generalizing multi-agent path finding for heterogeneous agents. In *Int. Symp. on Combinatorial Search (SOCS)*, pages 101–105.

Barer, M., Sharon, G., Stern, R., and Felner, A. (2014). Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Int. Symp. on Combinatorial Search (SOCS)*.

Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. V. (2004). Solving transition independent decentralized markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc.

Boyarski, E., Felner, A., Harabor, D., Stuckey, P. J., Cohen, L., Li, J., and Koenig, S. (2020). Iterative-deepening conflict-based search. In *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 4084–4090.

Boyarski, E., Felner, A., Sharon, G., and Stern, R. (2015a). Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding. In *Int. Conf. Automated Planning and Scheduling (ICAPS)*, pages 47–51.

Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., and Shimony, S. E. (2015b). ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 740–746.

Brafman, R. I. and Domshlak, C. (2008). From one to many: planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, pages 28–35.

Burkard, R. E., Dell'Amico, M., and Martello, S. (2009). *Assignment Problems*. SIAM.

Choudhury, S., Solovey, K., Kochenderfer, M. J., and Pavone, M. (2020). Efficient large-scale multi-drone delivery using transit networks. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 4543–4550.

Cil, I. and Mala, M. (2010). A multi-agent architecture for modelling and simulation of small military unit combat in asymmetric warfare. *Expert Systems with Application*, 37(2):1331–1343.

Coltin, B. and Veloso, M. M. (2014). Online pickup and delivery planning with transfers for mobile robots. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 5786–5791.

Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R. (2016). Analysis and observations from the first Amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188.

Deb, S., Yeddanapudi, M., Pattipati, K., and Bar-Shalom, Y. (1997). A generalized sd assignment algorithm for multisensor-multitarget state estimation. *IEEE Transactions on Aerospace and Electronic systems*, 33(2):523–538.

Dresner, K. and Stone, P. (2008). A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research (JAIR)*, 31:591–656.

Felner, A., Li, J., Boyarski, E., Ma, H., Cohen, L., Kumar, T. K. S., and Koenig, S. (2018). Adding heuristics to conflict-based search for multi-agent path finding. In *Int. Conf. Automated Planning and Scheduling (ICAPS)*, pages 83–87.

Felner, A., Stern, R., Shimony, S. E., Boyarski, E., Goldenberg, M., Sharon, G., Sturtevant, N. R., Wagner, G., and Surynek, P. (2017). Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Int. Symp. on Combinatorial Search (SOCS)*, pages 29–37.

Fioretto, F., Pontelli, E., and Yeoh, W. (2018). Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698.

Gebser, M., Obermeier, P., Otto, T., Schaub, T., Sabuncu, O., Nguyen, V., and Son, T. C. (2018). Experimenting with robotic intra-logistics domains. *Theory and Practice of Logic Programming (TPLP)*, 18(3-4):502–519.

Guestrin, C., Koller, D., and Parr, R. (2002). Multiagent planning with factored mdps. In *Advances in neural information processing systems*, pages 1523–1530.

Hönig, W., Kiesel, S., Tinka, A., Durham, J. W., and Ayanian, N. (2018). Conflict-based search with optimal task assignment. In *Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 757–765.

Hönig, W., Kumar, T. K. S., Cohen, L., Ma, H., Xu, H., Ayanian, N., and Koenig, S. (2016). Multi-agent path finding with kinematic constraints. In *Int. Conf. Automated Planning and Scheduling (ICAPS)*, pages 477–485. AAAI Press.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103.

Kitano, H. and Tadokoro, S. (2001). Robocup rescue: A grand challenge for multiagent and intelligent systems. *Artificial Intelligence*, 22(1):39–52.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, MA.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Kumar, A. and Zilberstein, S. (2009). Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*.

Larrosa, J. and Dechter, R. (2003). Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Constraints*, 8(3):303–326.

Li, J., Gange, G., Harabor, D., Stuckey, P. J., Ma, H., and Koenig, S. (2020). New techniques for pairwise symmetry breaking in multi-agent path finding. In *Int. Conf. Automated Planning and Scheduling (ICAPS)*, pages 193–201.

Li, J., Harabor, D., Stuckey, P. J., and Koenig, S. (2021). Pairwise symmetry reasoning for multi-agent path finding search. *Computing Research Repository (CoRR)*, abs/2103.07116.

Li, J., Harabor, D., Stuckey, P. J., Ma, H., and Koenig, S. (2019). Disjoint splitting for multi-agent path finding with conflict-based search. In *Int. Conf. Automated Planning and Scheduling (ICAPS)*, pages 279–283.

Liu, M., Ma, H., Li, J., and Koenig, S. (2019). Task and path planning for multi-agent pickup and delivery. In *Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1152–1160.

Loeliger, H.-A. (2004). An introduction to factor graphs. *IEEE Signal Processing Magazine*, 21(1):28–41.

Ma, H., Harabor, D., Stuckey, P. J., Li, J., and Koenig, S. (2019). Searching with consistent prioritization for multi-agent path finding. In *AAAI Conf. on Artificial Intelligence*, pages 7643–7650.

Ma, H. and Koenig, S. (2016). Optimal target assignment and path finding for teams of agents. In *Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1144–1152.

Ma, H., Koenig, S., Ayanian, N., Cohen, L., Hönig, W., Kumar, T. K. S., Uras, T., Xu, H., Tovey, C. A., and Sharon, G. (2017a). Overview: Generalizations of multi-agent path finding to real-world scenarios. *Computing Research Repository (CoRR)*, abs/1702.05515.

Ma, H., Li, J., Kumar, T. K. S., and Koenig, S. (2017b). Lifelong multi-agent path finding for online pickup and delivery tasks. In *Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 837–845.

Ma, H., Tovey, C. A., Sharon, G., Kumar, T. K. S., and Koenig, S. (2016). Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *AAAI Conf. on Artificial Intelligence*, pages 3166–3173.

Melo, F. S. and Veloso, M. (2011). Decentralized MDPs with sparse interactions. *Artificial Intelligence*.

Murray, C. C. and Raj, R. (2020). The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones. *Transportation Research Part C: Emerging Technologies*, 110:368–398.

Murty, K. G. (1968). Letter to the editor - an algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16(3):682–687.

Nair, R., Varakantham, P., Tambe, M., and Yokoo, M. (2005). Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *AAAI*, volume 5, pages 133–139.

Nissim, R., Brafman, R. I., and Domshlak, C. (2010). A general, fully distributed multi-agent planning algorithm. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*.

Oliehoek, F. A., Spaan, M. T., Vlassis, N., and Whiteson, S. (2008). Exploiting locality of interaction in factored dec-pomdps. In *Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 517–524.

Oliehoek, F. A., Whiteson, S., and Spaan, M. T. (2012). Exploiting structure in cooperative bayesian games. In *Uncertainty in Artificial Intelligence - Proceedings of the 28th Conference, UAI 2012*.

Popp, R. L., Pattipati, K. R., and Bar-Shalom, Y. (2001). m-best sd assignment algorithm with application to multitarget tracking. *IEEE Transactions on Aerospace and Electronic Systems*, 37(1):22–39.

Revach, G. (2018). Planning for cooperative multiple agents with sparse interactions. Master's thesis, Technion - Israel Institute of Technology, Haifa, IL.

Revach, G., Greshler, N., and Shimkin, N. (2020). Planning for cooperative multiple agents with sparse interaction constraints. In *The online Proceedings of the 6th Workshop on Distributed and Multi-Agent Planning (DMAP) at ICAPS 202*, pages 48–56.

Salzman, O. and Stern, R. (2020). Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1711–1715.

Scharpff, J., Roijers, D. M., Oliehoek, F. A., Spaan, M. T., de Weerdt, M. M., et al. (2016). Solving transition-independent multi-agent mdps with sparse interactions. In *AAAI*, pages 3174–3180.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2012a). Conflict-based search for optimal multi-agent path finding. In *AAAI Conf. on Artificial Intelligence*.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2012b). Meta-agent conflict-based search for optimal multi-agent path finding. In *Int. Symp. on Combinatorial Search (SOCS)*.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66.

Sharon, G., Stern, R., Goldenberg, M., and Felner, A. (2013). The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470–495.

Shome, R. (2021). Roadmaps for robot motion planning with groups of robots. *Current Robotics Reports*, pages 1–10.

Silver, D. (2005). Cooperative pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment*, pages 117–122.

Spieksma, F. C. (2000). Multi index assignment problems: complexity, approximation, applications. In *Nonlinear assignment problems*, pages 1–12. Springer.

Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Kumar, T. K. S., Barták, R., and Boyarski, E. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Int. Symp. on Combinatorial Search (SOCS)*, pages 151–159.

Sturtevant, N. (2012). Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2):144 – 148.

Surynek, P. (2020). Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering. *Computing Research Repository (CoRR)*, abs/2009.05161.

Švancara, J., Vlk, M., Stern, R., Atzmon, D., and Barták, R. (2019). Online multi-agent pathfinding. In *AAAI Conf. on Artificial Intelligence*, volume 33, pages 7732–7739.

Torreño, A., Onaindia, E., Komenda, A., and Stolba, M. (2017). Cooperative multi-agent planning: A survey. *Computing Research Repository (CoRR)*, abs/1711.09057.

Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *Artificial Intelligence*, 29(1):9–20.

Yu, J. and LaValle, S. M. (2012). Multi-agent path planning and network flow. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, volume 86, pages 157–173.

Yu, J. and Rus, D. (2014). Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, volume 107 of *Springer Tracts in Advanced Robotics*, pages 729–746.

בנוסף, אנו מתייחסים לחלק נוסף בבעיה – בעיית ההשמה אשר עוסקת בשאלה כיצד להקצות משימות לסוכנים בצורה מיטבית. אנו מציעים לנסח את בעיית ההשמה כבעיית אופטימיזציה עם אילוצים, משתמשים באלגוריתם קיים על מנת לפתור אותה בצורה מקורבת, ומציגים מספר שיטות על מנת לשלב את פתרון בעיית ההשמה בבעיית תכנון המסלולים.

על מנת לבחון את הביצועים של האלגוריתם המוצע והשיפורים השונים, אנו מציגים תוצאות של ניסוי אמפירי מקיף, הכולל בחינה של האלגוריתם בתרחישים שונים. תרחישי הבחינה בהם אנו משתמשים, משמשים גם לבחינת אלגוריתמים לפתרון בעיית התכנון הקלאסי, ובצורה זו אנו בוחנים את יעילות האלגוריתם, ואת יכולתו לפתור בעיות הולכות וגדלות.

בסיכום העבודה, אנו מציעים מספר הרחבות ושיפורים לאלגוריתם, ומספר הרחבות חשובות למודל של בעיית התכנון השיתופי, אשר תאפשרנה לייצג בעיות נוספות בעולם האמיתי.

תקציר

בעבודת מחקר זו, אנו מציגים וחוקרים את בעיית תכנון המסלול השיתופי לסוכנים מרובים (תיקרא מעתה בקיצור "בעיית התכנון השיתופי"). בעיה זו מהווה הרחבה לבעיית תכנון המסלול לסוכנים מרובים (תיקרא מעתה בקיצור "בעיית התכנון הקלאסית"), ומוסיפה עליה התנהגות שיתופית בין הסוכנים בצורה מפורשת.

בעיית התכנון הקלאסית עוסקת בקבוצה של סוכנים אוטונומיים אשר כולם פועלים ונעים בסביבה משותפת. בעיה זו היא שיתופית במהותה, שכן על כל סוכן בקבוצה להגיע מנקודת מוצא מסוימת לנקודת יעד מסוימת, מבלי להתנגש ביתר הסוכנים הקבוצה. אולם, בבעיות רבות בעולם האמיתי, סוכנים אשר פועלים בסביבה משותפת הם הטרוגניים, במובן שלכל סוכן ישנם סט שונה של יכולות ומגבלות. לפיכך, במסגרת בעיית התכנון השיתופי המוצגת בעבודה זו, היכולת של סוכנים להגיע ליעדם ולהשלים את משימותיהם, אינה תלויה רק במניעת התנגשויות בין סוכנים, אלא גם בתיאום (קואורדינציה) של החלטות ופעולות הסוכנים השונים. במילים פשוטות, אנו מעוניינים שסוכנים לא רק "לא יפריעו" אחד לשני, אלא יעזרו זה לזה בצורה מפורשת על מנת להשיג את מטרתם. אנו מכנים זאת תרחיש שיתופי אמיתי.

בתרחיש זה, קבוצה של סוכנים אוטונומיים פועלים בסביבה משותפת, ועליהם להשלים משימות שיתופיות על ידי שיתוף פעולה, ותוך שהם נמנעים מהתנגשויות עם סוכנים אחרים בקבוצה. על מנת להשלים משימות שיתופיות, על הסוכנים לשתף פעולה ולתאם את החלטותיהם ברמה גבוהה יותר מעבר למניעת התנגשויות בלבד. בעיה זו טומנת בחובה סיבוך חישובי מורכב, מעבר לסיבוך החישובי הקיים בבעיית התכנון הקלאסית, בו אנו נדרשים למצוא מסלולים לכל סוכן בקבוצה, ולמנוע התנגשויות בין הסוכנים השונים.

ההרחבה המוצעת בבעיית התכנון השיתופי משמשת מודל טבעי לבעיות רבות בעולם האמיתי, בהן סוכנים מסוגים שונים נדרשים לשתף פעולה כדי להשלים משימות. לפיכך, אנו מציעים מודל מתמטי לייצג את בעיית התכנון השיתופי, אשר מבוסס על מודל של בעיית התכנון הקלאסי. במודל המוצע , קיימים שני סוגים של סוכנים, וכל זוג סוכנים (אחד מכל סוג) עובד בשיתוף פעולה על משימה אחת. שיתוף הפעולה בין שני הסוכנים בא לידי ביטוי בקביעת פגישות ביניהם (מקום וזמן), ושואב מוטיבציה מתרחיש של רובוטים במחסן אשר מעבירים חבילה מאחד לשני. רק על ידי העברת החבילה בין הסוכנים, באפשרותם להשלים את המשימה.

בהתבסס על מודל זה, אנו מציעים אלגוריתם "חיפוש שיתופי מבוסס קונפליקטים", המבוסס על אלגוריתם קודם בשם "חיפוש מבוסס קונפליקטים", אשר פותר את בעיית התכנון הקלאסי בצורה אופטימאלית. האלגוריתם המוצע פותר בצורה אופטימאלית סט רחב מאוד של בעיות תכנון שיתופי. הוא משתמש במודול מיוחד לתכנון שיתופי הפעולה בין הסוכנים, אשר משולב בתוך האלגוריתם המקורי ("תכנון מבוסס קונפליקטים"), ומפריד בין תכנון שיתופי הפעולה (קביעת נקודות מפגש לכל זוג סוכנים), לתכנון המסלולים עצמם ומניעת התנגשויות. האלגוריתם מחשב נקודת מפגש מיטבית לכל משימה, ומייצר נקודות מפגש נוספות לפי הצורך, על מנת לבצע חיפוש יעיל במרחב הפתרונות. כדי לשפר את פעולת אלגוריתם התכנון השיתופי, אנו מציעים שני שיפורים, אשר מורידים את זמני הריצה שלו ובכך מעלים את אחוז התרחישים אותם הוא מצליח לפתור במסגרת זמן נתונה.

המחקר בוצע בהנחייתם של פרופ' נחום שימקין מהפקולטה להנדסת חשמל, וד"ר אורן זלצמן מהפקולטה למדעי המחשב.

חלק מן התוצאות בחיבור זה פורסמו כמאמרים מאת המחבר ושותפיו למחקר בכנסים ובכתבי-עת במהלך תקופת מחקר הדוקטורט של המחבר, אשר גרסאותיהם העדכניות ביותר הינן:

Nir Greshler, Ofir Gordon, Oren Salzman, and Nahum Shimkin. Cooperative multi-agent path finding: Beyond path planning and collision avoidance. Accpeted to the 3rd IEEE International Symposium on Multi-Robot and Multi-Agent Systems (MRS), 2021.

המאמר הבא פורסם במהלך לימוד תואר המסטר, אך אינו מהווה חלק מעבודה זו:

Guy Revach, Nir Greshler, Nahum Shimkin. Planning for Cooperative Multiple Agents with Sparse Interaction Constraints. In he online Proceedings of the 6th Workshop on Distributed and Multi-Agent Planning (DMAP) at ICAPS, pages 48-56, 2020.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

תכנון מסלול שיתופי לסוכנים מרובים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים בהנדסת חשמל

ניר גרשלר

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
חשון התשפ"ב חיפה אוקטובר 2021

תכנון מסלול שיתופי לסוכנים מרובים

ניר גרשלר